AD-A016 662

A RESEARCH PROGRAM IN COMPUTER TECHNOLOGY

University of Southern California

Prepared for:

Defense Advanced Research Projects Agency

September 1975

ARPA ORDER NO. 2223

ISI/SR-75-3

ANNUAL TECHNICAL REPORT
May 1974 - June 1975

A Research Program in Computer Technology

prepared for the
Advanced Research Projects Agency

INFORMATION SCIENCES INSTITUTE

UNIVERSITY OF SOUTHERN CALIFORNIA

4676 Admiralty Way/Marina del Rey/California 90291
(213) 822-1511

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ISI/SR-75-3	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* A Research Program in Computer Technology, Annual Technical Report, May 1974-May 1975.		5. TYPE OF REPORT & PERIOD COVERED Annual Technical Report May 1974 - June 1975 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) ISI Research staff		8. CONTRACT OR GRANT NUMBER(s) DAHC 15 72 C 0308
9. PERFORMING ORGANIZATION NAME AND ADDRESS USC/Information Sciences Insitute 4676 Admiralty Way Marina del Rey, CA 90291		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order #2223 Program Code 3D30 & 3P10
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE September 1975 13. NUMBER OF PAGES ~~96~~ 99
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* --------		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16 DISTRIBUTION STATEMENT *(of this Report)*

This document is approved for public release and sale; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

----

18. SUPPLEMENTARY NOTES

----

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

1: interactive theorem proving, lemma generator, Pascal, program correctness, program verification, Reduce, symbolic executor, verification condition.

2: ARPANET, control memory, microprogrammed processor, microprogramming, microprogramming language, microvisor, MLP-900, operating systems, resource sharing, TENEX, time sharing, writable control memory. (OVER)

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This report summarizes the research performed by USC/Information Sciences Institute from 1 May 1974 to 30 June 1975. The research is aimed at applying computer science and technology to problem areas of high DoD/military impact. The ISI program consists of eight research areas: Program Verification - logical proof of program validity; Programming Research Instrument - development of a major time-shared microprogramming facility; Automatic Programming - the study of acquiring and using problem knowledge for program generation; Protection Analysis - methods of assessing the viability of security (OVER)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE • S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

19. KEY WORDS (continued)

3: automatic programming, domain-independent interactive system, natural language, nonprocedural language, nonprofessional computer users, problem solving, problem specification, process transformation, world knowledge.
4: access control, computer security, encapsulation, error analysis, error-driven evaluation, error patterns, evaluation methods, protection mechanisms, software security, verification.
5: computer terminals, interactive message service, office automation, nonprofessional computer users, terminal-based message service.
6: computer network, digital voice communication, network conferencing, packet-switched networks, secure voice transmission, signal processing, speech processing, vocoding.
7: document printing capability, network terminal, text printing, Xerox Graphics Printer.
8: ARPANET interface, computer network, KA/KI, PDP-10, PDP-11/40, resource allocation, TENEX, user quotas.

20. ABSTRACT (continued)

mechanisms of operating systems; Information Automation - development of a user-oriented message service for large scale military requirements; Network Secure Communication - work on low-bandwidth, secure voice transmission using an asynchronous packet-switched network, Special Projects - primarily further development of Xerox Graphics Printer facilities; ARPANET TENEX Service - operation of TENEX service and continuing development of advanced support equipment.

ARPA ORDER NO. 2223

ISI SR-75-3

ANNUAL TECHNICAL REPORT
May 1974 - June 1975

A Research Program in Computer Technology

prepared for the
Advanced Research Projects Agency

Effective date of contract
17 May 1972

Contract expiration date
30 June 1975

Amount of contract
$7,661,344

Principal investigator
Keith W. Uncapher
(213) 822-1511

Co-principal investigator
Thomas O. Ellis
(213) 822-1511

INFORMATION SCIENCES INSTITUTE
4676 Admiralty Way/Marina del Rey/California 90291
(213) 822-1511

UNIVERSITY OF SOUTHERN CALIFORNIA

THIS RESEARCH IS SUPPORTED BY THE ADVANCED RESEARCH PROJECTS AGENCY UNDER CONTRACT NO. DAHC15 72 C 0308, ARPA ORDER NO. 2223 PROGRAM CODE NO. 3D30 AND 3P10

PERSONNEL

Research Staff:
Robert M. Balzer
Raymond Bates
Richard Bisbey II
Thomas L. Boynton
Jim Carlstedt
Stephen L. Casner
Danny Cohen
Martin J. Cohen
E. Randolph Cole
Stephen D. Crocker
Ronald L. Currier
Thomas O. Ellis
Lawrence M. Fagen
Louis Gallenson
Joel Goldberg
Neil M. Goldman
Norton R. Greenfeld
John F. Heafner
James Koda
Ralph L. London
Richard C. Mandell
David R. Musser
Donald R. Oestreicher
Robert Parker
Paul Raveling
Jeff Rothenberg
Walter R. Ryder
Robert H. Stotz
Ron Tugender
Keith W. Uncapher
Dono Van-Mierop
John J. Vittal
David S. Wile

Consultants:
Nancy L. Bryan
Gerald J. Popek

System Staff:
Alan E. Algustyniak
R. Jacque Bruninga
George W. Dietrich
Glen W. Gauthier
Donald R. Lovelace
Raymond L. Mason
Marion McKinley Jr.
William H. Moore
Vernon W. Reynolds
Dale S. Russell

Support Staff:
Robert Blechen
Ralph W. Caldwell
Wanda N. Canillas
Dale M. Chase
Jeannette Christensen
Kathie Colegrove
Nancy Dechter
Oralio E. Garza
Judy Gustafson
Patricia A. Hagedorn
Delia A. Heilig
Chloe Holg
Rose L. Kattlove
Kyle P. Lemmons
G. Nelson Lucas
Jack M. Mann
Katie Patterson
Betty Randall
Rennie Simpson
Nancy Travis
Deborah C. Williams

Research Assistants:
John K. Kastner
Donald S. Lynn
Larry Miller
David Wilczynski
Martin D. Yonke

CONTENTS

ABSTRACT

This report summarizes the research performed by USC/Information Sciences Institute from 17 May 1974 to 30 June 1975. The research is aimed at applying computer science and technology to problem areas of high DoD/military impact.

The ISI program consists of eight research areas: ***Program Verification***--logical proof of program validity; ***Programming Research Instrument***--development of a major time-shared microprogramming facility; ***Automatic Programming***--the study of acquiring and using problem knowledge for program generation; ***Protection Analysis***--methods of assessing the viability of security mechanisms of operating systems; ***Information Automation***--development of a user-oriented message service for large-scale military requirements; ***Network Secure Communication***--work on low-bandwidth, secure voice transmission using an asynchronous packet-switched network; ***Special Projects***--further development of Xerox Graphics Printer facilities; and ***ARPANET TENEX Service***--operation of TENEX service and continuing development of advanced support equipment.

EXECUTIVE OVERVIEW

The Information Sciences Institute (ISI), a research unit of the University of Southern California's School of Engineering, was formed in May 1972 to perform research in the fields of computer and communications sciences with an emphasis on systems and applications. The Institute, located off-campus, has sufficient autonomy within the University structure to assure it the freedom required to identify and engage in significant research programs.

A close relationship is maintained with USC academic programs through active cooperation among the Institute, the School of Engineering, the Department of Electrical Engineering, and the Computer Science Department. Ph.D. thesis supervision is an integral part of ISI programs, as is active participation of research assistants supporting ISI projects. Also, participating faculty and graduate students from other departments provide interdisciplinary capabilities for ISI projects.

The uniqueness of ISI is expressed in the following objectives defined at its founding:

- A major university-based computer science research center.
- A center with a largely full-time staff of researchers, augmented by graduate students and faculty.
- A center which possesses a unique blend of basic research talent and application and system expertise. The last two attributes are of special significance to the application of computer science and technology to key military problems.
- A university-based research center with strong active ties to the U.S. military community and a strong leadership role in identifying key computer R&D requirements in support of long-term military needs.

The Institute is structured to provide research and development capability at the system level--often required to assure an understanding of real problems and to provide useful solutions in transferable form. The management structure is such that virtually any researcher is known by the IPTO Director and Program Managers. Project

leaders share visibly in the responsibility for the conduct of each project and for the quality and impact of the research. At the end of the third year of operation, the full-time professional research staff numbers 37. The total number of ISI employees--including full-time research staff, participating faculty and graduate students, and support personnel--is 78.

The activities of ISI's eight major areas of research and associated support projects are summarized briefly below. Some of the research projects reported in this document are discrete activities in themselves; others can be seen as parts of a larger whole. For example, Program Verification, Automatic Programming, and the Programming Research Instrument projects should be considered as individual parts of an overall research effort in Programming Methodology; Information Automation, Network Secure Communication, and Special Projects are linked elements of a major investigation into Network Communications Technology. These mutual interdependencies among the various projects at ISI contribute largely to the fruitfulness of the Institute's research activities.

Program Verification. The goal of program verification research at ISI is to develop an effective program verification system for proving that computer programs are consistent with precisely stated detailed specifications of what the programs are intended to do. The system is expected to replace significant parts of testing in current software development, and will also provide important tools for judging the success of new programming language designs, new programming methodologies, and new detailed specification techniques. Already running at ISI is an initial, experimental version of an interactive program verification system based on the conventional inductive assertion method. The design philosophy is to provide automatic assistance for the verification process where practical, and otherwise to rely on human interaction. The system has verified numerous example programs. New capabilities and extensions are proposed which will permit the verification of a far wider class of programs than is now possible. The eventual impact will be an increase in the quality of software with an accompanying decrease in the cost of producing high-quality software.

Programming Research Instrument. This project has completed a highly reliable interactive microprogramming facility to be used as a general-purpose emulation laboratory for creating, manipulating, and debugging arbitrary computer architectures and high-level language processors. It consists of a powerful sharable microprogrammable CPU (the MLP) closely coupled to a TENEX system and appropriate software to allow interactive access to, and control of, the environment to be emulated via the ARPANET. The PRIM project personnel will integrate PRIM into the NSW architecture and aid NSW users as well as several other military or laboratory users in their introduction to and use of this facility.

Automatic Programming. The major effort of the Automatic Programming project is simply to allow users who are not computer programmers to functionally specify their application directly to a computer system, with the system transforming this input into a precise functional specification of the application. This system is intended to be both independent of any particular problem domain and able to deal with "loose" (i.e., incomplete, inconsistent, etc.) problem-oriented descriptions of a domain through a dialogue with the user. From this dialogue the system can acquire the "physics" (the objects, laws, relationships, etc.) of the loosely-defined domain, structure it, and use it to understand further communication and finally to write a program to accomplish the user's tasks. The system is being developed in the context of a (simplified) real-world problem, i.e., the militarily significant domain of first-level message distribution. It is currently able to acquire a domain description from the problem statement. The project was terminated at the end of the reporting period.

Protection Analysis. The goal of this project is to develop efficient techniques and semiautomated tools for detecting in operating systems various types of protection errors, i.e., errors that allow the systems to be compromised. The approach is empirical, based on the observations that (1) protection errors fall into a limited number of distinct classes and (2) "error patterns" representing the classes are effective criteria for finding the errors themselves. The method is to collect a data base of known errors, use it to determine the error classes, and (for each class) generate the appropriate error pattern and search algorithm. To date, errors from a variety of systems have been collected and a prototype package for finding errors of a single class has been built. The project proposes to extend the set of error classes and to provide packages for finding twelve different classes of errors.

Information Automation. The Information Automation project has a dual goal: 1) to develop the technology for providing on-line computer services directly to untrained users and 2) to develop a secure, on-line, interactive writer-to-reader message service for the military community. Such an on-line message service, new to the military, provides interactive assistance for formal messages from the initial draft preparation through coordination, transmission, and distribution. In addition, it will provide informal secure "off-the-record" communication without the need for face-to-face meetings. The message service is being developed in phases: the first addresses message preparation, the second will provide message delivery and reception, the third is concerned with debugging and test preparation, and the fourth is an operational test in a real military environment. Phase one is currently underway. A set of reports describing the design approach was produced in the fall of 1974 and an informal design was presented in January. Coding of the message preparation system is now in progress. As a tool to assist the system designer, a command language protocol analysis was developed and tested.

Network Secure Communication. The major objective of ARPA's Network Secure Communication project is to develop secure, high-quality, low-bandwidth, real-time, two-way digital voice communication over packet-switched computer communication networks. This kind of communication is a very high priority military goal for all levels of command and control activities. ISI's role in this effort is to continue developing the Network Voice Protocol required for communication of coded speech over a packet-switched network in real-time; to develop on-line voice conferencing capabilities; to continue implementation of the PDP11/SPS41 system for real-time LPC vocoding; and to develop dynamic off-line voice systems for storage and retrieval of voice files. During the recent months the ISI NSC project worked on (1) improving the quality of the real-time LPC system, implemented on the PDP-11/SPS41 system, (2) working toward getting the SPS hardware to be more reliable at ISI, and all the other ARPA sites, (3) modifying the existing PDP-11 operating system ("ELF") in order to achieve an efficient operational state, and (4) issuing the exact definition of the network voice conferencing protocol (NVCP).

Special Projects. The major Special Projects effort for the current year was the further development of ISI's and ARPA's Xerox Graphics Printer (XGP), a high-quality document printing capability in the form of a network terminal.

ARPANET TENEX Service. ISI is supporting, operating, and maintaining three complete TENEX systems on a schedule of 161 hr/wk each, in order both to provide TENEX service to ARPA and to support its research projects via the facilities at ISI. The Institute provides 24-hour availability of TENEX systems, maintenance, and operators; continued development/improvement support; and proper support of the XGP at IPTO. Through this support we have achieved increased long-term up-time; faster repair and improve preventive maintenance; economy of scale in operation; and the benefits of ISI expertise in establishing requirements for optimal loading and high reliability. In addition, this experience is used to assist in improving system reliability and to improve the number of users which can be handled with required response time.

PROGRAM VERIFICATION

Research Staff: *Ralph L. London*

Raymond L. Bates
Martin J. Cohen
Stephen D. Crocker
Lawrence M. Fagan
David R. Musser

Research Assistants: *Martin D. Yonke*
Donald S. Lynn

Support Staff: *Betty Randall*

GOALS AND IMPACT OF PROGRAM VERIFICATION

In many computer application areas the consequences of a program not performing as intended can be quite costly or damaging. The goal of program verification research at ISI is to develop a prototype program verification system for proving that programs are consistent with precisely stated detailed specifications. With such a system one will be able to achieve significant confidence that computer programs will perform as intended. This system will be an important part of finding solutions to the manifest problems of current software systems--their high cost, their unreliable behavior, the difficulty of modifying them, etc. [1]. The system will be used to help certify that software is correct and is expected to replace significant parts of testing in current software development. The system may be used in some cases to help determine whether protection and security specifications are met. The immediate impact will be a system that will, at last, permit programmers to demonstrate that their programs meet specifications. The system will also provide important tools for judging the success of new programming language designs, new programming methodologies, and new detailed specification techniques. The eventual result of advances in program verification will be an increase in the quality of software with an accompanying decrease in the cost of producing high quality software.

CURRENT ACCOMPLISHMENTS--A RUNNING SYSTEM

We have now produced at ISI an initial, experimental version of an interactive program verification system [2,3]. The design philosophy of the system is based on our strong belief that large parts of the total proof of actual programs can, and should, be done automatically, but also that in the foreseeable future some parts will have to be done by humans assisting the system. This seems a proper response to the genuinely open-ended nature of facts, theorems, and deductions needed to verify realistic programs. Thus our design strategy has been to provide automatic capability for the proof process where practical and to rely on human interaction for manual intervention otherwise. If a program can be verified with no human assistance, then we shall applaud the system's achievement. We expect, however, that the system will provide sufficient assistance so that the verification can be completed with minimal human hints or proof steps.

The main unique features of this system are its good facilities for user interaction, the modular system design which uses several previously existing components, the particular natural deduction theorem prover that is used, and the theorem prover's method of incremental bounding of variables, which, among other things, facilitates automatic proof by cases. The potential for modifying and expanding this system is an important feature, too.

The ISI program verification system has successfully verified numerous example programs, including binary search, various sorters, array rearrangement, arithmetic computations, parts of a prime sieve, a few routines in the verification system itself, and parts of two Lisp compilers. Of special encouragement and promise is the system's ability to support some abstractions both in the program and in its detailed specifications. This permits the proof to be completed as a structure of interconnected and intellectually manageable pieces, the only feasible way to verify large, complex programs. In other words, the aim is to prove many small pieces of a program and then to combine these proofs to verify large programs. Verifying large programs as single entities is doomed to failure.

A BRIEF DESCRIPTION OF THE SYSTEM

The ISI program verification system is based on the conventional inductive assertion method of proving properties of programs [4]. The verification task is decomposed into parts as follows: from the program and the detailed specifications, first produce a set of mathematical lemmas called verification conditions. The syntax and semantics of the programming language, which may be defined in several ways, are used in this step.

The goal is now to prove all of the lemmas; if successful, the program is verified. The proving starts by invoking various simplification and substitution capabilities (axioms, conditional transformations, and subgoaling) covering ordinary arithmetic, the problem domain, and the specifications.

Simplification alone often proves many of the lemmas. The remaining unproved lemmas are passed to the interactive theorem prover, where numerous theorem-proving capabilities are invoked. If unproved lemmas still remain, they must be analyzed (currently by humans) for several situations: (1) to see if a proof seems possible, perhaps by supplying hints or additional information to the theorem prover; (2) to see if the lemma is false, perhaps by constructing a counterexample, thereby indicating the need for changes to the program, to the specifications, or to both; or (3) perhaps the truth or falsity of the lemma cannot be determined, which indicates changes as in item 2.

The verification system consists of five major components: a standard text editor, a program and assertion parser for Pascal programs, a verification condition generator, a simplification and substitution package, and an interactive theorem prover. The entire system is Lisp-based, is now completely compiled, and runs as a large program on a PDP-10 computer. The system is implemented in Reduce, a Lisp-based symbolic mathematical system developed by A. C. Hearn. The particular Lisp that is used is UCI Lisp, primarily because of its impressive debugging facilities.

One of the important features of this system is the extent to which we have been able to use previously written and highly developed programs as major system components. First of all, Reduce, in addition to its powerful, well-developed algebraic manipulation capability, has served as an effective language for system implementation, and will permit the system to be as portable as Reduce itself. Our PDP-10 implementation of Reduce also has a built-in link to a text editor, and this provides the editor for the verification system. The Pascal parser, developed at ISI, was written in Reduce. The verification condition generator is essentially the Pascal generator of Igarashi, London, and Luckham [5], originally developed at Stanford. The simplification and substitution package was developed at ISI, drawing in part on the algebraic manipulation capability of Reduce. Several orders of magnitude in speed improvement have been achieved over some of our earlier simplification capabilities.

Another experimental simplifier, called CEVAL, is operational. CEVAL is an evaluator/simplifier of logical, relational, and arithmetic expressions. It has built-in knowledge of propositional calculus, the equality relation, and order relations and arithmetic operations on expressions representing integers and rational numbers. Additional domain-dependent knowledge is accessed via pattern match rules, obtained from a library of rules or supplied by the user. Simplification of arithmetic expressions

and pattern matching are handled by calls to the standard evaluator of the Reduce system. Conditional expressions (if-then-else) are used internally to represent all propositional calculus operators (not, and, or, implies, equivalent). CEVAL implements only a small set of transformations on conditional expressions, similar to those discussed by McCarthy [6] and implemented previously in the Boyer-Moore Theorem Prover [7]. However, it is still "complete" with respect to propositional calculus, in that any valid formula in the propositional calculus (i.e., provable by truth table) will be reduced to TRUE by the transformations. Output is available either in terms of the conditional operator or re-expressed in terms of not, and, or, and implies. Conditional expressions have also been used in pattern match rules to express axioms for abstract data types, as in Zilles [8] and Guttag [9]. Data types which have been axiomatized include arrays, stacks, queues, lists, sets, graphs, trees, and files. The convenience of conditional expressions in this usage was one of the motivations for choosing conditional expressions as the basic internal form of expressions in CEVAL.

The theorem prover is a variation of the prover described by Bledsoe and Bruell [10]. The prover originally was developed at the University of Texas at Austin in UT-Lisp, and was translated into Reduce for incorporation into the system. Although the prover has been modified to make it more effective on the types of theorems encountered in proving programs, its basic structure and interactive philosophy remain valid and unchallenged. We have recently added rational arithmetic to existing integer capabilities as well as additional interactive commands to allow the prover to work on specific subgoals and cases.

Currently, the normal mode of using the system is to invoke interactively the following system operations under the direction and assistance of an overseer program: create the program and and specifications, parse them checking for syntax errors, generate verification conditions, simplify them, and prove those that do not simplify to TRUE. The overseer includes important bookkeeping, checkpointing (dump and restore), and flexible proof step reordering facilities. The user can descend directly into Reduce or Lisp.

Users of the program verification system are expected to be skilled in both programming and in the problem domain for which they are writing programs. However, even with this expertise, users have a right to expect good human-factors features in the verification system. While much remains to be done in this important area, the verification system already does a credible job of displaying programs, theorems, specifications, formulas, the progress of a verification, user options, etc., in terms that are natural for humans. In particular, on TV-like terminals there are simple, but effective, special input/output facilities available, including split-screening.

A DETAILED EXAMPLE--BINARY SEARCH

Below is a transcript of the interactive verification system successfully verifying a binary search program. The transcript shows the dialogue that produced the verification. The actual proof is a small subset of this dialogue. Some of the output has been eliminated or slightly modified to save space without changing the sense of the human interaction or the steps of the actual verification. An example of successful verification of programs involving procedures and functions may be found in [2].

Conventions: The prompt for human input is ">"; thus human input follows the >. "$" asks for the default continuation at that point. Explanatory comments are enclosed in braces { }.

```
%This program does binary search on the array A[1 .. P-1] trying
%to locate the element X. If successful, then LOOKUP is set
%such that A[LOOKUP]=X and ERROR is set FALSE. If unsuccessful,
%ERROR is set TRUE.

ENTRY 1 < P & SORTED(A) & A[1] LE X & X < A[P];
EXIT (A[LOOKUP]=X) AND (ERROR=FALSE) OR NOTFOUND(X,A) AND ( ERROR =TRUE);
BEGIN M:=1;N:=P;ERROR:=FALSE;
ASSERT M < N & A[M] LE X & X < A[N] & SORTED (A) &ERROR=FALSE;
WHILE M+1<N DO BEGIN
      I:=(M+N)DIV 2;
      IF X < A[I] THEN N:=I ELSE IF A[I] < X THEN M:=I
            ELSE BEGIN LOOKUP:=I;GOTO 1 END
      END;
IF A[M] NE X THEN GO TO 2 ELSE BEGIN LOOKUP:=M;GO TO 1 END;
2: ASSERT NOTFOUND(X,A);ERROR:=TRUE;
1: ASSERT (A[LOOKUP]=X) AND (ERROR=FALSE) OR NOTFOUND(X,A) AND (ERROR=TRUE);
END.

>PROVE BSRCH;  {Start work on problem named BSRCH}
Parse: $ (FILE = BSRCH.PAS), RESET (FILE)
>$  Parsing file BSRCH.PAS     {the above program}
Vcgen: $ (UNIT = MAIN), RESET (UNIT)
>$using unit MAIN       {selecting main program over any
                    procedures / functions}
Generating verification conditions MAIN#1 ... MAIN#8
 Trying to simplify MAIN#1
Choice:  $ (PROCEED), +/-N, END, ASSUME, DEFER, STATUS, REDUCE
>$proceeding
        1<P
    and SORTED(A)
```

```
      and A[1] LE X
      and X < A[P]
 imp      1<P
      and A[1] LE X
      and X < A[P]
      and SORTED(A)
      and FALSE=FALSE
simplified to
TRUE
VC was MAIN#1
 Trying to simplify MAIN#2
Choice:  $ (PROCEED) ....
>$proceeding
         M<N
      and A[M] LE X
      and X < A[N]
      and SORTED(A)
      and ERROR=FALSE
      and M+1 < N
 imp      X < A[(M+N) DIV 2]
      imp      (M < (M+N) DIV 2) and (A[M] LE X)
           and X < A[(M+N) DIV 2]
           and SORTED(A)
           and ERROR=FALSE
simplified to
         X < A[(M+N) DIV 2]
      and X < A[N]
      and A[M] LE X
      and M+2 LE N
      and M<N
      and SORTED(A)
 imp M < (M+N) DIV 2
VC was MAIN#2     save as? >$MAIN#S2
 Trying to prove MAIN#S2
Choice:  $ (PROCEED) ....
>$proceeding
(SETUP)          { (x y z) denotes theorem prover operations;
                 here processing inequalities}
>Print Current theorem
         N in [M+2..POSINF]
      and M in [NEGINF..N-2]
      and A[M] in [NEGINF..X]
      and A[N] in (X..POSINF]
      and     X
           in [A[M]..
              MIN(A[(M+N) DIV 2] - 1,
                  A[N] - 1)]
```

```
      and A[(M+N) DIV 2] in (X..POSINF]
      and SORTED(A)
 imp M < (M+N) DIV 2
>$proceeding
...more time? >NO
M < (M+N) DIV 2 {conclusion of current subgoal}
Failed Time Limit
>USE
Lemma:
>2 * X LE Y IMP  X LE Y DIV 2;
save as? >DIVFACT
    X.*2 LE Y. {periods added to denote matchable variables}
 imp X. LE Y. DIV 2
[Confirm]
(SETUP USE-LEMMA)
>GO     {requests automatic completion of proof step}
(SETUP USE-LEMMA GO BACK-CHAIN)
[Using Y. := M+N ; X. := M+1]
M+2 LE N
PROVED by typelist
M < (M+N) DIV 2
PROVED
MAIN#S2 PROVED
 Trying to simplify MAIN#3
Choice:  S (PROCEED) ....
>$proceeding
        M<N
      and A[M] LE X
      and X < A[N]
      and SORTED(A)
      and ERROR=FALSE
      and M+i < N
 imp      NOT (X < A[(M+N) DIV 2])
     imp      A[(M+N) DIV 2] < X
         imp      (M+N) DIV 2 < N
              and A[(M+N) DIV 2] LE X
              and X < A[N]
              and SORTED(A)
              and ERROR=FALSE
simplified to
        X < A[N]
      and A[(M+N) DIV 2] < X
      and A[(M+N) DIV 2] LE X
      and A[M] LE X
      and M+2 LE N
      and M<N
      and SORTED(A)
```

```
 imp (M+N) DIV 2 < N
VC was MAIN*3     save as? >$MAIN*S3
 Trying to prove MAIN*S3
Choice:  $ (PROCEED) ....
>$proceeding
(SETUP)
>USE DIVFACT;  {the same lemma used in MAIN*S2}
Lemma:
    X.*2 LE Y.
 imp X. LE Y. DIV 2
[Confirm]
(SETUP USE-LEMMA)
>GO
(SETUP USE-LEMMA GO BACK-CHAIN)
[Using Y. := -(M+N) ; X. := 1-N]
M+2 LE N
PROVED by typelist
(M+N) DIV 2 < N
PROVED
MAIN*S3 PROVED
 Trying to simplify MAIN*4
Choice.  $ (PROCEED) ....
>$proceeding
        M<N
     and A[M] LE X
     and X < A[N]
     and SORTED(A)
     and ERROR=FALSE
     and M+1 < N
 imp     NOT (X < A[(M+N) DIV 2])
     imp     NOT (A[(M+N) DIV 2] < X)
         imp         A[(M+N) DIV 2] = X
                 and ERROR=FALSE
             or NOTFOUND(X, A) and (ERROR=TRUE)
simplified to
        X LE A[(M+N) DIV 2]
     and X < A[N]
     and A[(M+N) DIV 2] LE X
     and A[M] LE X
     and M+2 LE N
     and M<N
     and SORTED(A)
 imp    (FALSE=TRUE) and NOTFOUND(X, A)
     or X = A[(M+N) DIV 2]
VC was MAIN*4     save as? >$MAIN*S4
 Trying to prove MAIN*S4
Choice:  $ (PROCEED) ....
```

```
>Sproceeding
(SETUP)
>GO
(SETUP GO)
Typelist equality added to hypotheses: X = A[(M+N) DIV 2]
Disjunct found in hypotheses: X = A[(M+N) DIV 2]
   (FALSE=TRUE) and NOTFOUND(X, A)
 or X = A[(M+N) DIV 2]
PROVED
MAIN#S4 PROVED
 Trying to simplify MAIN#5
Choice:  S (PROCEED) ....
>Sproceeding
         M<N
      and A[M] LE X
      and X < A[N]
      and SORTED(A)
      and ERROR=FALSE
      and NOT (M+1 < N)
 imp A[M] NE X imp NOTFOUND(X, A)
simplified to
         X < A[N]
      and A[M] LE X
      and M<N
      and N LE M+1
      and X NE A[M]
      and SORTED(A)
 imp NOTFOUND(X, A)
VC was MAIN#5     save as? >$MAIN#S5
 Trying to prove MAIN#S5
Choice:  S (PROCEED) ....
>Sproceeding
>GO
(GO SETUP)
Typelist equality added to hypotheses: N = M+1
Automatic substitution: M:=N-1
(GO SETUP =SUB)
..NEED MORE INFORMATION
>USE
Lemma:
>SORTED(A) AND ( A[K] < X) AND (X < A[K+1]) IMP NOTFOUND(X,A);
save as? >SORT1
         X. < A.[K.+1]
      and A.[K.] < X.
      and SORTED(A.)
 imp NOTFOUND(X., A.)
[Confirm]
```

```
(GO SETUP =SUB USE-LEMMA BACK-CHAIN)
[Using K. := N-1 ; X. := X ; A. := A]
X < A[N]
PROVED
A[N-1] < X
PROVED by typelist
    X. < A.[K.-1]
 and A.[K.] < X.
 and SORTED(A.)
PROVED
NOTFOUND(X, A)
PROVED
MAIN#S5 PROVED
 Trying to simplify MAIN#6
Choice:  3 (PROCEED) ....
>DEFER          {requests delayed action on this goal}
 Trying to simplify MAIN#7
Choice:  S (PROCEED) ....
>$proceeding
    NOTFOUND(X, A)
 imp     (A[LOOKUP] = X) and (TRUE=FALSE)
     or NOTFOUND(X, A) and (TRUE=TRUE)
simplified to
TRUE
VC was MAIN#7
 Trying to simplify MAIN#8
Choice:  S (PROCEED) ....
>$proceeding
        (A[LOOKUP] = X) and (ERROR=FALSE)
     or NOTFOUND(X, A) and (ERROR=TRUE)
 imp     (A[LOOKUP] = X) and (ERROR=FALSE)
     or NOTFOUND(X, A) and (ERROR=TRUE)
simplified to
TRUE
VC was MAIN#8
Prove: NO(S),UN(DEFERRED), DEF(ERRED) (VC'S), or (RE)VCGEN
>DEF            {requests action on deferred goals}
 Trying to simplify MAIN#6
Choice:  S (PROCEED) ....
>$proceeding
        M<N
     and A[M] LE X
     and X < A[N]
     and SORTED(A)
     and ERROR=FALSE
     and NOT (M+1 < N)
 imp      NOT (A[M] NE X)
```

```
    imp     (A[M] = X) and (ERROR=FALSE)
        or NOTFOUND(X, A) and (ERROR=TRUE)
Proposed substitution: X := A[M]
>YES
Sub used: X:=A[M]
simplified to
TRUE
VC was MAIN#6
>STATUS;
MAIN#1 ==>        PROVED by simplifier
MAIN#2 ==> MAIN#S2 PROVED by prover
MAIN#3 ==> MAIN#S3 PROVED by prover
MAIN#4 ==> MAIN#S4 PROVED by prover
MAIN#5 ==> MAIN#S5 PROVED by prover
MAIN#6 ==>        PROVED by simplifier
MAIN#7 ==>        PROVED by simplifier
MAIN#8 ==>        PROVED by simplifier
```

OVERALL EXPECTED PROJECT ACHIEVEMENTS

Verifying programs is not now a trivial task, nor should one expect the task ever to become trivial. Some people claim that computer programs are among the most complex objects created by the human mind. Accordingly, while we are optimistic, we neither seek nor expect panaceas in the area of program verification. We do expect, however, effective systems to aid humans in the verification task (and indeed in the joint effort of program construction and verification).

The current ISI verification system is already useful in helping to verify programs, as shown by the some fifteen examples from the verification literature it has successfully verified. More importantly, by virtue of its generally modular construction, we can experiment with new system components and with new verification strategies and ideas. We will continue to do this as we make more of our verification experience available to others in the form of an effective program verification system. Yet, we are also very much aware of the system's shortcomings--both minor and fundamental. Correcting the minor limitations can be viewed as system tuning, and some of this remains to be done. Our major continuing efforts, however, will be directed to overcoming fundamental restrictions in achieving a truly effective system for widespread human use in verifying large, important programs. Success here will permit the verification, and hence improved quality, of a far wider class of programs involving many more programming language constructs. Furthermore, such verifications will be markedly more credible and much easier and less expensive to accomplish than is now possible. The payoff will be software that is known to meet specifications and to work as intended.

The user will ultimately see an improved version of the currently running system which has been described previously. He will present to the system both his program and his detailed specifications of what that program does. More precisely, the user will present the program part-by-part along with the corresponding specifications; the overall verification will be completed by combining the veritications of these parts. The system will keep track of the progress of the proof and, in particular, will indicate what remains to be proved. It will accept advice, hints, and changes from the user and will provide suggestions to the user. It will collect all the facts which the user advises may be assumed in the verification so that the user (and others) will know what assumptions were made in the verification. It will present the final verification in a readable form. The system will play the dual role of overseer of the verification and of helpful assistant for the user as both the user and the system jointly work on verifying the user's program. The user will thus be able to verify his programs or to discover where errors exist in either his program or his specifications. The main use of the system will be in achieving software that provably does what it is supposed to do. The system will also, as noted, be useful in judging the success of new programming language designs, new programming methodologies, and new detailed specification techniques.

As we develop interactive and automatic verification tools during the course of this project, we expect to use them successfully on significant, real computer programs which will be selected from both computer science and military applications. In this way we will successfully demonstrate that the goals, impacts, and payoffs that are described are indeed achievable.

REFERENCES

1. Goldberg, J., ed., ***Proceedings of a Symposium on the High Cost of Software,*** Monterey, California, September 1973. Published by Stanford Research Institute.

2. Good, D. I., London, R. L., and Bledsoe, W. W., "An interactive program verification system," ***Proceedings of International Conference on Reliable Software,*** April 1975, 482-492. Also, ***IEEE Transactions on Software Engineering,*** SE-1, 1, March 1975, 59-67.

3. London, R. L. and Musser, D. R., "The application of a symbolic mathematical system to program verification," ***Proceedings of ACM Annual Conference,*** 1974, 265-273.

4. Floyd, R. W., "Assigning meanings to programs," Proc. of a Symposium in Applied Mathematics, Vol. 19--***Mathematical Aspects of Computer Science,*** J. T. Schwartz, ed., American Mathematical Society, Providence, R.I., 1967, 19-32.

5. Igarashi, S., London, R. L., and Luckham, D. C., "Automatic program verification I: A logical basis and its implementation," ***Acta Informatica,*** 4, 2, 1975, 145-182.

6. McCarthy, J., "A basis for a mathematical theory of computation", ***Computer Programming and Formal Systems,*** P. Braffort and D. Hirschberg, eds., North Holland Publishing Company, Amsterdam, 1963, 33-70.

7. Boyer, R. S. and Moore, J S., "Proving theorems about LISP functions," ***J. ACM,*** 22, 1, January 1975, 129-144.

8. Zilles, S. N., "Algebraic specification of data types", ***Project MAC Progress Report*** 11, Massachusetts Institute of Technology, Cambridge, Mass., 1974.

9. Guttag, J. V., "Dyadic specification and its impact on reliability," ***Three Approaches to Reliable Software: Language Design, Dyadic Specification, Complimentary Semantics,*** J. Donahue, J. D. Gannon, J. V. Guttag, and J. J. Horning, Technical Report CSRG-45, Computer Systems Research Group, University of Toronto, Toronto, Canada, December 1974, 48-88.

10. Bledsoe, W. W. and Bruell, P., "A man-machine theorem-proving system," ***Advance Papers of Third International Joint Conference on Artificial Intelligence,*** 1973, 56-65. Also, ***Artificial Intelligence,*** 5, 1, Spring 1974, 51-72.

PROGRAMMING RESEARCH INSTRUMENT

Research Staff: *Louis Gallenson*

Raymond Bates
Joel Goldberg
Raymond L. Mason

Support Staff: *George W. Dietrich*
Oralio E. Garza
Rennie Simpson

DESCRIPTION

The PRIM (Programming Research Instrument) project has completed an interactive microprogramming facility to be used as a general-purpose emulation laboratory for creating, manipulating, and debugging arbitrary computer architectures and high-level language processors. A unique service on the ARPANET, it consists of a powerful sharable microprogrammable CPU (the MLP-900) closely coupled with a TENEX system and appropriate software to allow interactive access to, and control of, the computing environment the user wishes to emulate. The MLP-900 has proved to be reliable in continuous operation since August 1974, with the primary applications being the emulation of existing minicomputers for experimentation and evaluation. A library of emulators is being developed as user population grows; the library currently consists of a basic PDP-10 (developed as a test vehicle for the system), a PDP-8, a PDP-11, an AN/UYK-20, a Univac 1050 MOD2, and a Nova (CPU only). As the emulator library continues to grow, the PRIM facility should become more attractive to a larger user community. Current plans also include allowing users access to these virtual machines (emulators) via a National Software Works Tool Bearing Host. This will make available several new NSW tools with a single interface. (See Section 3 for a further discussion of the National Software Works (NSW).) PRIM is therefore becoming a service facility, providing a unique tool to groups of NSW programmers, as well as an experimental computer environment for the researcher.

THE PRIM FACILITY

The PRIM system was developed at ISI as a subsystem of TENEX, the time-sharing system of Bolt, Beranek and Newman, Inc. PRIM consists of the MLP-900 microprogrammable processor together with appropriate software to drive the MLP-900, to support MLP-900 microprogramming, and to provide an environment in which users create, manipulate, and interact with their emulators and/or emulated systems, and user documentation.

Hardware

PRIM's hardware system is based on two processors: the Digital Equipment Corporation's PDP-10 and the STANDARD Computer Corporation's MLP-900 prototype processor. (See Fig. 2.1.) The PDP-10 and MLP-900 share memory as dual processors; the MLP-900 is also a device on the PDP-10 I/O bus. The PDP-10, connected to the ARPANET, runs under TENEX with a paged virtual memory. Its processor contains 256K words of 36-bit memory. The I/O operations performed by TENEX include file, terminal, and network handling, swapping, and all other accesses to peripheral devices.

The MLP-900 is a vertical-word microprogrammed computer (microprocessor) that runs synchronously with a 4-MHz clock. It is characterized by two parallel computing engines: the Operating Engine (OE), which performs arithmetic operations, and the Control Engine (CE), which performs control operations. The OE contains 32 36-bit general-purpose registers for operands and 32 36-bit mask registers to specify operand fields. A 1K 36-bit high-speed auxiliary memory is associated with the OE. The CE contains 256 state flip-flops, a 16-word hardware subroutine return stack, and 16 8-bit pointer registers. The MLP-900 is accessible only through the PDP-10 as outlined above (i.e., the I/O bus and shared memory); no provisions have been made for direct connection of any peripheral devices. The introduction of a microvisor state has been of major importance to the PRIM project. Prior to this project, little had been done toward making the multitude of available microprogrammed processors potentially sharable resources. This initial experiment goes a long way toward making microprogrammed processors widely and inexpensively available.

Figure 2.1 The MLP-900.

Software

There are three principal items of PRIM software:

- The General Purpose Microprogramming Language (GPM) compiler.
- The MLP-900 microprogram supervisor (microvisor).
- The TENEX MLP-900 programs, i.e., the MLP-900 driver and MLP-EXEC.

The GPM compiler was essentially completed in early 1973; for a more detailed account of its development the reader should consult Ref. 1.

GPM and the GPM Compiler. GPM is a high-level machine-oriented language, written in TENEX BLISS, designed explicitly for the MLP-900. As a high-level language, GPM offers a block structure and statement syntax similar to PL/1 or ALGOL. The specific statement types defined in GPM are generalizations of the actual MLP-900 MINIFLOW instruction set; constructs completely foreign to MINIFLOW (e.g., multiplication) do not appear in GPM. As a simple example of MINIFLOW generalization, consider that the result of a GEAR (GEneral ARithmetic) ministep may be shifted left or right only by 0, 1, 2, 4, 6, 8, 12, or 16 bits; in GPM, any shift amount may be specified, and the compiler will generate multiple shifts as required.

As the production language for the MLP-900, GPM is constrained to satisfy many of the usual requirements of an assembly language. First, there is a well-defined subset of GPM statements that produce exactly one ministep per statement; the subset is capable of generating all possible ministeps. Second, multi-ministep statements do not generate implicit side effects; for example, a complex arithmetic assignment that requires a temporary register for an intermediate result will generate a compile-time error unless the programmer has explicitly declared some register to be available as a temporary.

The GPM compiler is successfully being used to write diagnostics for the MLP-900 and test user software (emulation of a PDP-10). Experience with the compiler reveals that minor modifications and suggested speed improvements may be required. The improvements will be considered as more measurement data is accumulated and specific critical code is further identified.

MLP-900 Microvisor. The MLP-900 microprogram supervisor (microvisor) is a small, fully protected resident system that controls the MLP-900 and its communication with the PDP-10. It loads and unloads the user's MLP-900 context upon command from the PDP-10, supports paging of the user target program, protects main memory and the rest of the PDP-10 system from user interpreter errors, and provides the interpreter with a few services, such as an extended subroutine stack and calls for external communication. (The microvisor requires 356 (octal) words of control memory, including its Action Request locations.)

The microvisor performs the functions normally expected of an operating system, the difference being that it is written in microcode and supervises the execution of microcode. The microvisor interacts only with the user microcode and the TENEX MLP driver; it does not provide any facilities for--or impose any restrictions upon--the user target system. User microcode is subject to the restrictions imposed by the user mode MLP-900 hardware.

PDP-10 Support Programs. The PDP-10 TENEX software for support of the MLP-900 consists of a driver to control communication with--and sharing of--the MLP-900, and a subsystem (MLP-EXEC) to allow interactive access to the MLP-900 for a user at a TENEX terminal. The MLP driver and its TENEX JSYS's comprise the interface to the MLP-900 used by MLP-EXEC.

The TENEX MLP-900 Driver. As mentioned above, access to the MLP-900 from a TENEX process is accomplished via the MLP driver in TENEX. The driver is the extension in TENEX of the microvisor; all communication with the MLP-900 goes through the driver. While new microcode "machines" can be designed and debugged under the MLP-EXEC, completed ones will work directly through their own terminal subsystems, which will communicate directly with the driver. Communication with the driver is accomplished through a series of JSYS's which mimic (roughly) the JSYS's for subsidiary fork control. The two principal elements involved in creating and running the MLP are the MLP context (the user microcode together with all the MLP registers) and the target system upon which the context is to operate. The calling process must build both before establishing access to the MLP.

The context is a structure that contains all the data necessary to load the MLP and begin (or resume) execution of the desired microcode. It includes not only an image of the MLP-900 control memory, but also the internal MLP-900 registers and some cells used by the driver to implement MLP-900 communication with the PDP-10. The context is 10 memory pages (5120 words) long, and must begin on a page boundary in the caller's address space.

The target system is the memory upon which the MLP context is to operate. It is defined as a TENEX fork (or process)--either the caller or a subsidiary fork established solely for this purpose. Typically, the target system fork (SFORK or SFRKV) will never be started on the PDP-10; it exists to define an address space for MLP execution.

To protect the ISI TENEX system and lessen the impact of MLP debugging (both hardware and software) the initial version of the driver has been implemented almost entirely as a normal user process rather than as part of the TENEX operating system. This preliminary driver is being used in debugging the entire system, including the interfaces between the microvisor and the driver, and between MLP-EXEC and the driver. While the differences between preliminary and final driver are transparent to both the microvisor and the user microcode, there are some unavoidable differences for the calling TENEX process. MLP-EXEC is aware of the differences, and handles them properly; to the user of MLP-EXEC, the only visible difference is that the response time is longer.

MLP-EXEC and Its Commands. MLP-EXEC is a user program, called via TENEX, written primarily in BLISS. The program basically consists of two modules: the I/O handler (which includes file access and target memory allocation) and the debugging facility (MLP DDT). The MLP-EXEC commands assume a familiarity with TENEX Exec commands; a subset of TENEX commands is implemented for functions similar to those of the TENEX Exec.

MLP-EXEC provides an environment in which the user at a terminal can compile, load, execute, and debug MLP-900 microcode in a manner similar to that used for debugging programs on the PDP-10. In addition, he can create and debug target programs and environments--although these tools must be provided at a very primitive level, since MLP-EXEC cannot know the nature of the target environment.

The MLP-EXEC "ready" character, ">," signals the user to enter a command. Commands to MLP-EXEC can specify any of several types of actions:

- Controlling the loading, execution, or debugging of the MLP context.
- Controlling the loading and debugging of the target system.
- Setting up the input/output files for the MLP.
- Providing access to the TENEX within MLP-EXEC as a convenience.

All the commands for user context manipulation begin with a period (".") These include LOAD, RESET, CONTINUE, RUN, SAVE, GET, and DDT commands.

All of the commands for the target system begin with the character "/" and use standard TENEX subsystems in responding to the command (i.e., /LOAD invokes the standard TENEX loader to load a relocatable binary file into the target system's address space). These include GET, MERGE, DDT, SAVE, SSAVE, and RESET commands.

The command format, key words, arguments, and separators are identical to those used in TENEX. MLP-EXEC prompts for each field required by the user's command, and the escape terminator will complete abbreviated commands. Additionally, two characters (Control T and Control C) act as commands in themselves to control MLP execution and to provide status information on the MLP. Editing control characters are also included to edit command key words and arguments.

User Interpreter and Target Program. The user's interpreter is a program written in GPM to run on the MLP-900; it defines a (re-entrant) MLP-900 control memory image. This image, together with all the nonprivileged registers and flip-flops within the MLP-900, is the MLP-900 context; user's contexts are loaded and unloaded as the MLP driver shares the MLP among different users.

The context defines the user's interpreter (or target machine) and operates upon the user target program in a totally arbitrary way. The only constraint upon the target program is that it fit into a 512K, 36-bit (virtual) memory space.

PAST EFFORTS

PRIM has been a major project at ISI since the inception of the Institute in May of 1972. The goals of the project have remained essentially the same throughout this period of time: to provide a flexible experimental computing environment available to the researcher via the ARPANET. The implementation of this unique facility successfully demonstrated several firsts:

1. To take advantage of a rich source of existing software, the microprogrammable computing engine was closely coupled to a TENEX system which provides all the I/O and file handling capabilities. We were able to create a microprogrammable environment and still minimize the new software required to provide the researchers with the needed tools.

2. The microprogrammable computer has been implemented as a multi-access sharable environment requiring an executive state. The MLP-900 hardware was modified to provide protection of the resident microvisor, as well as protection between users of PRIM and the TENEX systems.

3. A generalized debugger is currently being implemented to facilitate debugging of target programs for a variety of environments and emulated computer architectures.

The first year's effort was primarily concerned with system design, hardware development, and the design of a compiler to produce microcode for the MLP-900. The second year's effort completed the designs and implementations of the hardware and software requirements. This included integration and checkout of the hardware, diagnostics and the software to drive the MLP-900 from the TENEX operating system. The third year's effort was primarily concerned with interacting with potential users, developing emulators, final debugging, and subjectively monitoring the use and acceptance of the facility. The documentation of the facility was also completed: ***PRIM User's Manual*** [1] and the ***Maintenance Manual*** (the latter in draft form).

CURRENT EFFORTS

Recently we have been involved in introducing PRIM to a number of users in the military community to demonstrate its utility, evaluate its acceptance, and collect data for inputs to IPTO for planning management of this facility. We have taken two approaches in introducing the PRIM system. First, wide distribution (more than 200 copies) of the ***PRIM Overview***, coupled with invitations to an introductory workshop and demonstration of the system. Second, direct contact with some thirty individuals representing twenty organizations within the ARPA and military communities. Most of the direct contacts were initiated by us; the rest were the result of queries by persons receiving the ***Overview***.

Our approach is to seek out potential users within the military commands, assess their problems, and (where warranted) join with them in a team effort to solve these problems. A joint effort between ISI and NELC has produced an AN/UYK-20 emulator in

order to conduct a set of experiments required by an NELC Software Development Group. The AADC project at NADC has expressed interest in using PRIM for development work and computer architecture studies. As mentioned at the beginning of this section, ISI is also using PRIM as a Tool Bearing Host of the National Software Works. We also approached selected individuals in the academic community (University of Southern California, University of California at Los Angeles, and Carnegie-Mellon University), encouraging students with qualified projects to use the PRIM facility.

The Symbolic Manipulation of Computer Descriptions (SMCD) project at CMU will utilize a microprogrammable computer to achieve its goals and has completed a comparison study of two candidates.* The MLP-900 and a PDP-11/40E are two of the candidates for the computer. The PDP-11 is a 16-bit machine housing 1K word of 80 bit microstore, and basic cycle time of 140 to 300 nanoseconds. Four primary benchmark programs were written for each candidate including the Nova emulator (basic CPU). The PDP-11 is a somewhat faster target machine except where the user can take advantage of the wider data path of the MLP (36 bits) as in a multi-word integer multiply. The MLP is somewhat easier to program the primary project task, an optimizing micro-compiler, although the task is rated as difficult for both machines. These conclusions are consistent with the expected advantages of both types of microprogrammable computers, vectoral and parallel. The SMCD project will continue this evaluation, since many outside factors must be considered before selecting the optimal tool to satisfy their research needs. NELC continues to express strong interest in the PRIM facility. They are in the final stages of debugging their AN/UYK-20 emulator and are preparing a series of experiments for a dual host connection using the MLP-900 TENEX and second host on the ARPANET. NELC has generated a proposal to build and support a System Design Laboratory (SDL) - the core of which would be the MLP-900 - and to continue to provide this unique service to the development community via the ARPANET. Moving the MLP-900 to NELC would help to successfully terminate this project, finding an appropriate home for the MLP-900 in the military environment is one of the project's major goals for FY77.

Recent interest in the MLP-900 has been expressed by: ADTC at Eglin Air Force Base to evaluate micro-processor architecture, Dr. Wesley Chu of UCLA for a study and evaluation of bus architecture, and provide an instructional vehicle for a course in microprocessors, and NSW to satisfy the needs of a family of Tool Bearing Hosts.

The NSW requirements for a TBH are easily and economically satisfied by PRIM capabilities. These requirements (connectivity to the ARPANET, file handling, control of tasks (start, stop etc.,) and user accounting and status reports) are all available to NSW by upgrading the existing TENEX to a TENEX TBH operating system. New tools can then be added by building new emulators within PRIM; as the library of emulators grow, the MLP-900 will provide a family of NSW tools. The first of these emulated systems is a Univac 1050 MOD 2, which serves as a base computer in many Air Force installations. The instruction set (CPU) for this virtual machine has been completed and is currently

---

* Oakley, John, "A Comparison of Two Microprogrammable Processors: PDP-11/40E and MLP-900," Department of Computer Science, Carnegie-Mellon University, May 1975.

being debugged. The implementation of the I/O environment required to complete this task should be available by August 1975. We estimate 3 to 4 man-months for the average system emulation effort. On an average the emulated system will run 2 to 10 times slower than real time. For this compromise in speed, the economical advantage of building emulators over electrically interfacing system and debugging software as a host machine on the ARPANET is significant. (For new Host/IMP interfaces the average costs are $20,000 hardware and 4 to 6 man-months of programming and testing.) In the case of the U1050, a reasonable ARPANET Host interface is not economically feasible. Of greater advantage for the NSW environment is the capability of building emulators richer in debugging capability than the original systems, simplifying the user's programming tasks.

DEBUGGER

Work is proceeding on a new interactive debugging facility which allows a user--an emulator implementor or the user of an emulated machine--to inspect and/or modify his PRIM environment in a form which is understandable and comfortable to him. Implementors work in terms of MLP registers, control memory, target memory, and GPM code; user programmers in terms of their machine's memory, registers, and assembly language. (The implementor's facility is improved over the old debugging facility only to the extent that he can use the "user" facility when the mood strikes: most of the improvement is directed toward the user of the implemented target machines, who formerly operated in octal or a primitive subset of his assembly language.) In order that the debugger handle the various target machine languages, each machine implementor must now also generate the various tables and routines required by the debugger to "understand" his machine's assembly language. We anticipate that, at least for most minicomputers, the work to develop the necessary tables will be insignificant in relation to the work required to develop the emulator itself.

The primary goal of this work is the creation, for each emulated machine, of a familiar debugging environment with a minimum of new rules and strange syntactic forms. Due to the great diversity of machines-and more importantly, of assembly languages-the overall system is designed to be extremely flexible in its syntactic demands. A secondary goal of the work is the creation of a relatively simple--though not necessarily familiar--specification of the machine-specific items which must be encoded to add a new machine to the debugger's vocabulary. Adding a new machine to the system requires the cooperation of both the implementor and PRIM system personnel; the initial machine vocabulary, including AN/UYK-20, PDP-11, and Univac 1050, is being generated as part of the initial effort. To the extent that this secondary goal is in conflict with the first goal, the interface specification will suffer. Design work on the system is now completed, and implementation of the initial version is under way, with a scheduled release date of July 1975.

The MLP-900 continues to operate reliably and throughout FY75 we experienced only three hardware failures. The system has been available to the users whenever TENEX was available. Effort to locate and correct marginal operations, especially within the TENEX interface subsystem, continues, for we have experienced occasional

unexplained malfunctions. These malfunctions have been limited to main memory transfers, affecting the current user only, and do not require a system restart.

The operational reliability of the MLP-900 is stated with guarded optimism because to date we have had minimal use of the system. During the past year we estimate about 70 hours of MLP-900 CPU time with six different users running or debugging five different emulators. Continued reliable operation as the user population increases and more emulators become available would provide sufficient reliability data. We are optimistic that this will occur in FY76. However, the MLP-900 currently runs diagnostics two hours each evening in order to keep it exercised.

FUTURE EFFORTS

The goal for PRIM in FY76 is to evaluate its utility as a service on the ARPANET. To this end we will continue to work cooperatively with the aforementioned users, seek new users, and complete PRIM's transition from experimental testbed to a new service on the ARPANET.

At the start of this contract period, in FY76, PRIM will have the beginnings of a library of emulators: PDP-11, PDP-8, AN/UYK-20, U1050 and Nova. With this library of emulators (of minicomputers), PRIM can offer an additional service for users interested in experimenting with programs for minicomputers. With the power of TENEX (editors, compilers, debuggers, etc.) we can offer a significant amount more development power for the multitude of minicomputer users. We intend to explore and expand this use of PRIM throughout this proposal period. Similarly, we intend to aid programs such as the National Software Works to use PRIM as a special design tool available on the ARPANET.

A major portion of our future effort will be spent in building an MLP tool for NSW. A tool, in NSW, is a logical device capable of responding to commands of the user. Implementation of MLP tools will require rewriting of the MLP-EXEC to conform to NSW specifications. The new EXEC (MLP-TOOL) will provide a standard interface to all tools running on the MLP-900. The first of these tools being implemented is the UNIVAC 1050 MOD2. We are contemplating the need for at least two more, currently unspecified virtual tools.

Each of the tools developed for NSW will also require at least two levels of user interaction requiring additional software development within the MLP-TOOL. A user can specify a tool, i.e., U1050, either as a bare CPU or with operating system. In either case, the users next request will probably be to load a specified program. In one case the command will be interpreted by the MLP-TOOL and in the latter case by the target system being run by the MLP-900. The latter case will be the prominent one being used by NSW users, and the implementation task is one of making most of the loads, gets, configuring I/O environment invisible to the user. The PRIM user currently commands the system to perform all these tasks to provide the flexibility required by the researcher. The NSW user is typically a programmer trying to create and debug a program, and NSW provides the tools for facilitating this task.

Another program we intend to support is the NELC effort to design a System Design Laboratory, mentioned above. SDL is a tool for solving some of the problems associated with designing large systems in the Navy Laboratory Communities. The MLP-900 will provide some of the original capabilities required by SDL. Therefore the tentative plan is to move the MLP-900 to NELC in San Diego, California. Planning the move, training NELC personnel, and helping in design of the SDL could require a significant amount of PRIM project effort. SDL will continue to support all users via the ARPANET so that the MLP-900 users except for 3 to 4 months of nonavailability will not be aware of this move. This would bring the PRIM project to a successful conclusion by early FY77 with all major project goals being achieved: creation of the facility, using the facility in nontrivial ways and transferring the technology to users within the military community.

REFERENCE

1 ***PRIM User's Manual***, ISI/TM-75-1, April 1975.

AUTOMATIC PROGRAMMING

Research Staff: *Robert M. Balzer*

Neil M. Goldman
Norton R. Greenfeld
Walter R. Ryder
John J. Vittal
David S. Wile

Research Assistant: *David Wilczynski*

Support Staff: *Chloe Holg*

INTRODUCTION

The goal of ISI's Automatic Programming project is simply to allow experts of an application area who are not programmers to functionally specify their application directly to a computer system, with the system transforming this input into a precise operational functional specification of the application. Such an accomplishment represents a testable model of the proposed application which could be used as follows:

- As a precise specification of the desired application program from which a human programmer could generate the application and against which the implementation could be tested.

- To examine the functional behavior of the application against the requirements and, if necessary, to modify the functional specifications until they satisfy the requirements.

- As the input to an automatic test data generator which would develop test cases to comprehensively exercise the model.

Because DoD's activities are so diverse, such a system must be capable of accepting specifications for a wide variety of applications.

It is well known that software is in a desperate state. It is often unreliable, delivered late, unresponsive to change, inefficient, and expensive. Furthermore, since it is currently labor intensive, the situation will further deteriorate as demand increases

and labor costs rise. Thus DoD faces one of two choices: either increase the productivity of highly trained, carefully selected specialists or reduce the training requirements through automation, thereby broadening the base of qualified users. Structured programming, built around the concept of discipline, addresses the first path, the approach we propose the second. We feel that the first approach will perpetuate much of the current crisis as systems continue to become more complex. Only automating the process can control the enormous complexity, improve the reliability, modifiability, and efficiency, and reduce the cost. For this approach to be successful, the system must acquire and use a semantic description of a domain--a particular universe of discourse--to understand the user's statements, fill in omitted details, and maintain consistency.

APPROACH

Functionally, the two most important characteristics of our proposed system are its independence from any particular problem domain and its attempt to deal directly with nonprofessional computer users without the intervention of computer programmers--choices which have largely dictated the direction of the project. Domain independence requires that the domain "physics"--its objects and their relationships with other objects, its laws, its transformations, and its constraints--be available in a processable form within the system and that the system be general enough to deal effectively with a wide variety of such physics. Direct interaction with nonprofessional computer users means that both the physics and the problem statements will be in problem-oriented (as opposed to computer-oriented) terms, preferably in natural language, and that they will be "loose" descriptions containing incomplete, inconsistent, and irrelevant statements rather than a precise formal structure. The primary goal of our system is to acquire from a dialogue with the user the physics of the loosely defined domain, structure it, and use it to understand further communication specifying an application, to remove the imprecisions from the specifications, and finally to organize the separate pieces into an operational and testable specification which accomplishes the user's stated task.

The constraints and restrictions of the computer have increasingly been incorporated into programming advances for several years. They are manifest in better languages, automatic storage mechanisms, and optimizations of many forms. On the other hand, the structure, constraints, and limitations of the problem domain have generally not been incorporated into programming systems. A major theme of automatic programming (in fact the characteristic distinction between it and conventional programming) is the use of such knowledge--an issue which raises a number of questions. If the system is to understand something of a domain, how is the knowledge

on which this understanding is based to be represented? What procedures can be made available for exploiting this knowledge in guiding the system's interaction with a user and in generating programs? How, in particular, is the essentially nonprocedural information in constraints and limitations to be reflected in a procedural form? What can be done to help identify inconsistencies? How can the system be given a capacity for inference similar to that which forms the mainstay of human communication and which allows obvious details to be left unspecified? Will the system be able to understand its own products well enough to be able to modify them in response to changed requirements? Answers to these questions define the front on which important advances in specification acquisition and analysis will be made.

Our system contains four knowledge bases: knowledge of specifications and their structure (how specifications are organized from parts, what constitutes a well formed specification, etc.), knowledge about application specifications and their acquisition (what kinds of imprecise specifications are used, how the resolution of one imprecision may affect the resolution of another, etc.), knowledge about domain descriptions and their acquisition (what constitutes a well formed description, how information from such a description can be used to resolve an imprecision or affect the program organization, etc.), and finally knowledge about a specific domain (such as what objects exist, how they are related to other objects, what actions are performed, what constraints exist, etc.). Of these four knowledge bases, the first three are fixed. Only the last, the knowledge of a particular domain, is acquired and changes from problem to problem. This is feasible because the domain description knowledge base defines the form of the acquired description--basically as a universe of types and interrelationships between them. It is through such a simple model that we can concentrate on imprecision removal and program organization rather than confusing these with the specifics of a particular domain.

To concentrate on this knowledge extraction and domain structuring activity, we have assumed the existence of a natural language parser which transforms the user's input into a parsed case structure. Such a parser is currently beyond the state of the art, but this goal is actively being pursued by other groups and we expect it to be available by the time our project is ready to assemble a total system. Until then, we are manually transforming the natural language input into the case-structured form required. If such a parser does not materialize, we would have to use a more restrictive and formal subset of natural language.

As a second means of limiting the scope of our work, we have decided to omit efficiency concerns for the operational specifications generated; we have focused on generating a logically correct operational specification for the user's needs without attempting to optimize it. This has greatly simplified our effort by allowing us to

directly model the user's domain in a data-representation-free manner through an associative data base, and hence obtain operational specifications which are much closer to the user's conception of the problem. By not having to introduce extraneous details (such as data representation) during the construction phase, we have been able to concentrate on the specification's logical behavior. Furthermore, we firmly believe that such representation-free and behaviorly specified descriptions are the correct way to specify applications and that optimization should occur during a separate and later implementation phase (not part of this project). It is clear that such an approach would greatly simplify the maintenance problem. The logical-behavior specification would be modified and then reoptimized through reimplementation.

We are also limiting the range and type of domains allowed. In addition to size limitations, these domains are characterized by such features as a rich structure of types, no parallelism, no second or higher-order constraints or inference rules, and no time dependencies other than ordering.

The final and most important means of controlling the scope of the project and thereby assuring orderly progress are the types and amounts of imprecision allowed in the input. If no imprecision of any type is allowed, then the input is already in a formal programming language and though no work is required to output an operational specification, the notion of dealing with nonprogrammers has completely disappeared and all that has been produced is the design of a better (or worse) program specification language. On the other hand, if no restrictions are placed on the imprecisions allowed in the input, the task clearly becomes infeasible. Between these two extremes can be determined an allowed level for each type of imprecision which so significantly simplifies program specification that nonprogrammers can deal directly with the system while keeping the task of removing these imprecisions manageable.

Two valid scientific questions have been raised by our approach. The first questions the viability of a domain-independent approach at this time. The concern is that not enough is known about domain descriptions and how they differ to build a domain-independent system. We feel that by attempting domain independence we force ourselves to address the issues of domain structure and of obtaining guidelines from this structure without allowing a particular structure to become embedded in our system. Also, by removing representations from the specification, we have removed a great deal of dependence on the particular domain structures.

The second question assumes that the first has been answered and therefore that a domain-independent system such as ours can generate unoptimized representation-free programs. It questions whether automatic optimization of such programs is possible. This is primarily an issue of data representation selection. Other research groups, most

notably Rochester, are concerned with these issues. They are, however, primarily concerned with implementation representation of logical structures such as sets or lists. A higher order representation selection issue exists which addresses the logical aggregation of data--what items should be collected together to form a record, what aggregations should be treated as sets, how they are accessed and ordered, etc. Such work is unfortunately not currently being investigated. However, three of the four possible benefits of our project are independent of such work. Only the use of our output specification directly as a practical programming language requires the automatic representation selection problem to be solved.

SPECIFICS OF THE APPROACH

We are building a system with two major components: Domain Acquisition and Model Completion. The former sequentially processes a set of statements describing the user's problem and the domain in which it exists. It is responsible for extracting from these statements the description of the object being manipulated, the actions performed on them, the criteria necessary and sufficient to perform these actions, the constraints which must be satisfied, and the rules for inferring information not explicitly stated. This information may be given directly, may be inferred from example usage, or may be assumed in order to make sense of the input. Some of this information may have been previously acquired and saved in a domain description.

This component is implemented through a production system in which each transformation rule has a pattern which, if found in the input, activates the rule. An activated rule will typically assert some extracted knowledge in the associative data base and rewrite the input with the extracted information omitted or transformed. This activation process is continued until no rule matches the (transformed) input. Then the next input is processed.

A production schema was chosen because of its orientation toward case analysis, its facility for expanding as new rules are added, and its ability to accept manual transformations for unimplemented rules (see the Accomplishments subsection).

During these transformations, when an ambiguous interpretation is noted, one of three actions is taken: the problem can be kept for later processing in the hope that new information will resolve the ambiguity; the user can be directly asked to resolve the ambiguity; or the system can establish a backtracking point, assume one interpretation, and be prepared to back up and assume the other. Currently, only the first two options are used, since our system has no backtracking capability.

The Model Completion component is responsible for all interstatement processing. Its main function is to form a precise operational specification by organizing the actions referenced in the individual statements into an appropriate control structure. These actions are organized into sequential segments or asynchronously activated daemons in a two-stage process. First, the needs, requirements, and results of each action are analyzed to determine any implicit ordering restrictions. This partial ordering is then merged with any explicit partial ordering specified in the input to produce the final ordering restrictions. The second stage determines which actions should be treated as asynchronous daemons and removes them from the ordering. It then attempts to find a total ordering consistent with the restrictions. Finally, all action descriptions, action invocations, and object references are transformed into an executable form.

RELATION TO OTHER WORK

Our project is related to other similar work primarily at MIT, Stanford, and IBM Yorktown. The MIT effort is aimed at producing highly optimized programs for a specific domain through built-in knowledge of the general methods and techniques of the domain particularized to the user's needs. Thus, although both our domain descriptions are similar, we focus on acquiring this description and determining logical behavior; they, on the other hand, take these as given and are concerned with particularizing them and choosing an optimized method. The work at IBM is very similar in approach to that of MIT, though for a very different domain. It also differs from the latter in that it conceives of two systems, the first of which is a programming language designed specifically for the domain which embodies the current knowledge of model particularization. When this system is completed, a user will be able to specify his problem as a specialization of the domain and be questioned by the system for the particulars. This will enable the efficient generation of users' programs, although the language and interactions are formalized. The second IBM system attempts to use a natural language interface to loosen this rigidity.

Both of these efforts, like our own, are aimed at nonprogrammers. The Stanford project, however, is aimed at reducing the amount of detail that a programmer must specify to generate a program. It is thus concerned with different methods of specifying an algorithm, such as by input/output pairs, traces, predicates, and/or description. It is also concerned with programmer areas such as list processing and sorting rather than user domains, as are the other projects.

ACCOMPLISHMENTS

1. ***Semi-automatic acquisition of a real-world example domain.*** Our system, with the help of a small number of manual transformations (to be removed later as implementation proceeds), accepts a parsed version of a loose natural language description of a real-world domain and extracts a processible description of the domain in terms of the objects, their relationship with one another, the actions they participate in, and the constraints they must follow.

2. ***Acquisition of types, events, and relations.*** As part of the above domain acquisition, the system identifies examples of types, events, and relations in the input and builds descriptions of them. For instance, use of a relation or event (verb) with new cases (English keywords identifying an argument) and/or argument types causes a more generalized description of the relation or event to be constructed.

3. ***Discovery and implementation of the "knowledge acquisition" heuristic.*** When a relation instance is asserted to be true, the relation may have several preconditions associated with it which, if not satisfied, indicate a contradiction. If one or more are known to be false, then a true contradiction exists and the assertion is not allowed. However, if a precondition can be neither proved nor disputed--which is the most common case when new information is being acquired--then the "knowledge acquisition" heuristic states that it should be assumed true. This heuristic enables the system to acquire information "by side effect" through additional structure necessary to support explicit information.

4. ***Requirement analysis and input/output determination.*** An implemented part of Model Completion analyzes the requirements and results of each event (currently hand-generated) to determine a partial ordering of the events necessary to ensure that requirements are produced before being required. This partial ordering will eventually be merged with explicit ordering statements from the user to produce an ordering space within which a total ordering for the generated program can be found. This analysis also determines input and output data for the program by finding, respectively, the data used but not produced, and the data produced but not used.

5. ***AP1.*** Our AP system is based on an associative relational data base. The assertion and retrieval mechanisms necessary for such a data base have been incorporated into AP1--an extension to INTERLISP. AP1 also includes mechanisms for maintaining the consistency of the data base through constraints and for implicit data through inference rules applied when required data is not found explicitly. AP1 is also intended as the language in which we generate programs so that they can also utilize an associative relational data base. Through a reimplementation of the data base, an order-of-magnitude improvement in speed was achieved.

6. ***Transformation debugger.*** We have implemented and are using an interactive debugger for production systems which raises to the transformation level debugging facilities normally found only at the language level. It records the action associated with each transformation, enables conditional breakpoints to be inserted in or between transformations, enables manual transformations to be built and remembered, and eventually will allow back up to previous states. This greatly facilitates our style of example-driven constructions of the transformation rules. That is, rules are built only as needed, but are made more general than the specific example demands.

7. ***Programmer's interface.*** While looking for a suitable implementation language, we designed and built a mechanism for transforming interactive programming languages into programming systems by utilizing the programming tools and environment already constructed for INTERLISP, although this was not directly related to project goals. While useful in itself as a means of increasing programmer productivity, this work had its real significance as a model for ARPA's National Software Works project.

CURRENT STATUS OF PROJECT

The Automatic Programming project was terminated at the end of the current reporting period. It will be superseded by the Specification Acquisition from Experts (SAFE) project, whose plans are as outlined below.

We have decided to develop our system in the context of a real-world (albeit simplified) problem. We selected the militarily significant domain of first-level message distribution, and have extracted from an Army functional specifications manual a short, simplified, and very high level loose description of an implemented system.

With the help of some manual transformations this description has been processed and analyzed by the Domain Acquisition component. The Model Completion component is largely unimplemented, but (as mentioned earlier) one part which takes the requirements and results of the actions described and produces the implicit partial ordering is working. Furthermore, it identifies the inputs and outputs of the system by finding, respectively, the information used but never produced and the information produced but never used.

In FY76 we plan to finish the message distribution example and to select and present to our system three different real-world domains of approximately the same size and complexity as the message distribution domain. Such domains are characterized by:

a. Natural language specification of about one typewritten page for the combination of domain and problem (this must be manually transformed into our actual input language).

b. Input specification must adhere to system's imprecision conventions.

c. Input specification must adhere to system's domain restriction conventions (such as no parallel control structure, no second order constraints or inference rules, etc.).

d. Not more than 25 object types, 15 relations on those object types, 10 actions, and 100 to 150 total instances of all objects (first 3 restrictions limit size of domain which must be understood, while last restriction limits size of domain which must be executed and is designed to prevent overloading either the associative memory or LISP memory).

As we address new domains, more transformations will become necessary to handle new situations previously unencountered. The new transformations may interfere with the existing ones. We will have to identify and resolve such conflicts.

The main goal of these studies will be to determine the generality of our system in terms of the amount of overlap, and the amount of conflict, with existing facilities. In some sense, we must develop an estimation of the size of the "vocabulary" (i.e., the facilities) needed to handle domain descriptions. We will also be studying how to specify a domain and application and how to represent them in the system.

This understanding of domain and application descriptions will allow us to accept more imprecise and incomplete specifications by resolving or filling in information from information specified elsewhere and through knowledge of domain structures and interrelationships. We will continue to push on this front until we can handle specifications typically found in functional specification manuals.

If we were totally successful in attaining domain independence, then new domains could be accepted without any modification of the system by merely providing their domain description. We do not expect to achieve such a level of independence. However, our goal is to minimize such modification so that by the end of FY76 we can acquire and handle a new domain of roughly the size and complexity of the message-distribution domain in less than a week.

In FY77, while continuing to develop the facilities described above, our main focus will be on the sizing issues raised by dealing with large unsimplified real world problems. These problems of enlarging size arise with almost all aspects of the problem: the problem specification and vocabulary, the number of ambiguities and number of interactions with the user, and the possible interactions between the specified actions, etc.

Dealing with large domains is critical, however, if the project effort is ever to pay off. In large programming efforts, communication problems between team members abound, compatibility and consistency are of paramount importance, and the complexity is overwhelming for any individual. This situation is tailor-made for automation of specification acquisition, analysis, consistency and completeness validation if we can provide a system which scales up from current laboratory demonstrations.

Assuming that the system has been sized to handle significant real-world problems, in FY78 we would like to work with a military user to acquire, analyze, and debug the specification of a task currently being implemented within the military. Since the program specification produced would be highly inefficient, we would expect it only to be used to test that the specification matched the user's intention, that is, that the specified system behavior matched the desired behavior. This is not normally the case in real projects where the number of errors and inconsistencies in a specification usually exceeds those produced in the implementation. Having a testable specification would go a long way in reducing such design errors before implementation began. Furthermore, as a precise operational specification of the problem, the specification itself or natural language paraphrase of it could be used to resolve for the human programmers any ambiguities in the original specification.

PROTECTION ANALYSIS

Research Staff: *Richard Bisbey II*
Jim Carlstedt

Consultant: *Gerald J. Popek*

Support Staff: *Betty L. Randall*

THE PROBLEM BEING SOLVED

During the past decade some computer manufacturers have claimed that certain of their general-purpose operating systems (i.e., systems with generalized information-sharing facilities) were secure and could be used to store, process, and protect classified or sensitive information. Unfortunately, these claims are overly optimistic; a general-purpose operating system that is secure against malicious attack does not actually exist in either the commercial sector or the research community. The problem lies partly in both faulty design and faulty implementation. While long-term research is progressing in the design of secure systems and the verification of software, the computer user will not feel its impact for several years. Currently there are more than 6000 computer systems in the DoD and its contractor facilities, covered by the DoD Industrial Security Program. Many of these facilities require resource sharing at multiple levels of security, but cannot achieve it because of operating system vulnerabilities. (By "vulnerability" we mean a protection error that allows the integrity of the system itself--and thus its protection mechanism--to be compromised. System integrity is obviously the most critical aspect of operating system security, since without it the protection policies of the system with respect to the integrity and privacy of the user's data cannot be assured of correct or complete enforcement.) The goal of this research is to help remove these vulnerabilities by developing efficient techniques and automatable tools for detecting them. There is very clear evidence in the military regarding the lack of multi-level security at the operating system level. The cost to the military in not having such security is high in dollar expenditures and in risks to classified information.

In what follows, the terms "techniques" and "tools" are used to denote error-finding aids developed by the project. The former denotes general methods or strategies,

while the latter denotes procedures for applying these methods or strategies. The procedures are expressed in varying degrees of formality, and are not written in any particular programming language. Computer programs that implement them are necessarily specific to particular operating systems and thus are the responsibility of the users of these tools.

RELATIONS TO OTHER WORK

The work described here is only one of a number of existing or potential efforts to improve the security of operating systems. Perhaps the best way to indicate this project's relationship to other work is by means of a "subject tree" showing its position in the total field of computer security. (See Fig. 4.1.)

COMPUTER SECURITY

Physical installation
Operations and maintenance personnel
Communication facilities
Storage facilities
OPERATING SYSTEMS (PROTECTION)
    Theoretical studies
    Design of new systems and mechanisms
    ENHANCEMENT OF EXISTING SYSTEMS
        Design modification
        ERROR DETECTION
            Formal verification
            INFORMAL METHODS
                Dynamic methods (penetration testing, auditing)
                STATIC METHODS
                    Evaluation activities
                    TOOL DEVELOPMENT
                        Simple debugging techniques
                        ADVANCED AUTOMATED AND SEMIAUTOMATED TECHNIQUES

Figure 4.1

This is only a rough taxonomy: there are many other ways to categorize work in the security field, and many projects include work in more than one category. Nevertheless, it does indicate areas of actual and potential work and the relationships between them.

The distinction between operating system security and other aspects of computer security needs no explanation here. There is much current activity in this area, some of it concerned primarily with attempts to gain a better understanding of basic problems and possibilities (e.g., the work at Rutgers University), some with the design of new systems incorporating more advanced protection schemes (e.g., the work at Carnegie-Mellon University, the University of California at Berkeley, and UCLA), and some with the enhancement of current operating systems. The latter category consists of both redesign work whose goal is essentially the same as that of new-system design (e.g., the effort at MIT to identify a minimal security kernel for Multics) and work concerned with the problem of error detection.

Error detection methods can be classified first of all as either formal or informal. In the former category is verification of software in general via "proofs of correctness" (e.g., the work in this area at ISI), with verification of operating systems presenting special problems (being studied at Stanford Research Institute). Informal methods, typified by traditional debugging techniques, fall into two major categories: (1) dynamic, involving the execution of the target program or system, and (2) static, relying primarily on the analysis of program listings and system documentation. Dynamic techniques have been widely used to find protection errors, either by including special auditing routines in the system itself or by employing "blind" penetration techniques (e.g., earlier work at System Development Corporation).

A penetration attempt is more likely to succeed if it employs static as well as dynamic techniques (e.g., earlier work at Rand and ISI). When the object is to produce a more general evaluation of the security of an operating system, a large-scale penetration effort may be launched to find as many errors as possible in a target system (e.g., current work at System Development Corporation and Lawrence Livermore Laboratory). Currently, such error detection activities rely more on the organization and expertise of their personnel than on the effectiveness of their tools. This is an expensive and, we believe, unnecessary situation. For this reason, ISI's Protection Analysis project is engaged in an effort to develop more effective evaluation techniques and tools, primarily those that can be used in the static mode by people with less knowledge of security considerations.

APPROACH

The approach taken is an empirical one, based on two observations:

1. Protection errors fall into distinct classes or "types." Errors of the same type appear many times, not only in functionally different portions of the same operating system, but in different operating systems as well. Furthermore, there is reason to believe that the number of error types representing vulnerabilities in operating systems is finite and small--probably not more than 25. This is based on an initial analysis of a number of errors from a variety of operating systems including OS/360, GCOS, Multics, Exec-8, and TENEX and error categorization efforts by IBM [1] and others [2]; it is supported by the proposition that system integrity depends on a quite limited number of design requirements.

2. "Error patterns" representing error types can be used as effective criteria for searching for protection errors of those types. We have experienced, and witnessed in others, a large difference in effectiveness between a "blind search" and a search for errors of a particular well-described type. We have observed that even persons with no previous experience in protection analysis can find protection errors when given a specific error pattern to guide their search.

The approach is thus twofold: (1) to derive the error types and (2) to generate for each error type a tool or technique that can be applied in an automated or semiautomated fashion to find protection errors that are instances of that error type. Error patterns are the common vehicle for the approach.

To derive the error types, descriptions of protection errors are initially converted into error patterns by listing the minimal set of conditions that constituted the original error. Each pattern is generalized by substituting more generic or abstract features for their more specific counterparts and by deleting qualifying details, both without affecting the essence of the conditions themselves. This process results in a hierarchy with the most general and abstractly described patterns at the upper levels and the most specialized and concrete ones at the lower levels. The converse of generalization is "instantiation," where a pattern is transformed by substituting for more general features the more specific counterparts that occur in particular classes of operating systems or particular functional areas, resulting in a more concrete pattern.

Conceptually, an error pattern forms the reference input to an automated or semiautomated error detection algorithm that is general-purpose in the sense that it can be used to find instances of the corresponding error type in any of a large class of operating systems. Prerequisite to automating such an algorithm is the definition of a "comparison language" whose objects are the features of the pattern and whose structure and notation are suitable for expressing relevant portions of the target system. The error detection process thus consists of two steps: (1) the "normalization" of the operating system by filtering out irrelevant features and mapping the relevant features into the comparison language and (2) searching the normalized representation for feature combinations matching the given pattern. The first step is system-dependent, but can be partially automated in at least some cases. The second step is system-independent and in principle can be fully automated. For convenience and relative simplicity, the choice of a comparison language and the design of normalization and comparison algorithms are done on a pattern-by-pattern basis.

PROGRESS

Work on pattern-based protection evaluation was begun in October 1973. The following has been accomplished:

1. ***Project design.*** The need and potential utility of such a project were evaluated, the approach was developed, and the major tasks--pattern data base development and design of normalization and error detection algorithms--were defined in terms of processes, information flow, and associated problems.

2. ***Collection and analysis of protection errors.*** Informal descriptions of protection errors from a variety of operating systems, including OS/360, GCOS, Multics, Exec-8, and TENEX, were collected and initial versions of the corresponding first level-patterns were generated for these errors. Several of the first level-patterns have been generalized.

3. ***Feasibility test.*** To test the feasibility of the approach, an experiment was conducted in which a single pattern was applied to portions of the Multics operating system. The experiment was carried out as follows: An error type was selected for which pattern features could be fairly easily described. A set of guidelines were stated for recognizing instances of this pattern. The guidelines were applied manually to several microfiche listings of Multics source code, resulting in the discovery of several security errors.

4. ***Prototype error detection packages.*** A computer program was written to automate much of the error detection process described in item 3. Source copies of portions of the Multics operating system obtained via the ARPANET were processed by the program, and the output was manually examined for errors. Previously unknown security errors were found. A second prototype package for discovering errors of a different type has been built and is being tested.

5. ***Reporting.*** A research report describing the approach was prepared [3], together with a detailed report describing the first error type and prototype programs used in the Multics experiment [4].

RESEARCH AND DEVELOPMENT

The Protection Analysis project will continue the development of effective, economical, and reliable detection techniques and tools for security errors in operating system software. As indicated above, this work falls into three categories:

1. ***Error collection and analysis.*** The purpose of this activity is to extend the pattern data base "horizontally," in order both to identify new variations of existing patterns and to increase the potential coverage of the techniques developed with respect to the error types to which they may be applied.

2. ***Pattern analysis and data base development.*** The data base must be extended not only "horizontally" but also "vertically" by generalizing existing patterns to the more abstract levels at which interpattern relationships are more easily recognized and at which error types can be most effectively identified and represented. While this requires careful analysis, it yields insights valuable to the invention of error detection algorithms, simultaneously improving the quality of the patterns themselves. Expansion of the error pattern data base will necessitate the continued development and refinement of a formal notation to express patterns in terms of both the protection policy and the error conditions.

3. ***Algorithm development.*** As indicated earlier, this is actually a set of activities, one for each error type. Our plan is to concentrate on that type or those types for which the payoff/cost appears to be greatest, where payoff is estimated in terms of the "exploitability" of errors of this type in existing operating systems and where cost is a measure of the difficulties anticipated in algorithm development. Depending on the error type, the difficulties here can be substantial; the conditions of the search pattern

must be expressed in terms of static features that can be recognized in a system description, rather than dynamic features that have representations only in an executing system. There is, moreover, little precedent to draw on for techniques of operating system normalization. Fortunately, by relaxing somewhat the conditions of a pattern, it appears that normalization difficulties can sometimes be eased considerably at the expense of some additional screening of error detection output.

4. ***Transfer documentation.*** Much of the documentation will originate during the development activities themselves. However, a final set of user guidelines will be required when the development of a prototype has been effectively completed. We also include in this category interim reports on error types and patterns, detection techniques, system fixes, design implications, and general protection insights.

IMPACT

The work described here will have an impact in several areas. Most immediate, of course, is the impact on evaluation activities for existing operating systems with respect to the reliability of their security mechanisms. The empirical basis of the technique makes it easy to incorporate new error types as they are identified and as algorithms for them can be developed. The techniques can thus be used also in computer acquisition as one of a set of standard tests which must be met for system acceptance. Continuing back up the "subject tree" displayed earlier, we can anticipate impacts at every level. As seen in the Multics example [4], static pattern-directed tools can also suggest corresponding dynamic error-detection techniques. By negating conditions found in error patterns, assertions can be identified for use in formal verification of the protection aspects of operating systems. Along the same line, the data base of error patterns is useful in the repair or modification of existing systems; since a pattern is the minimal set of conditions that must hold for an error of that type to be present, repair is simply the negation of at least one of the conditions. Similarly, the pattern data base forms the basis for a "best practices manual," i.e., a list of errors that should be avoided in the design of new systems and protection mechanisms. Finally, the analysis needed to derive error patterns, to generalize them to abstract levels, and to develop associated error detection algorithms yields insights that contribute to a deeper understanding of protection itself.

REFERENCES

1. McPhee, W. S., "Operating System Integrity in OS/VS2," ***IBM Systems Journal***, Vol. 13, No. 3, 1974, pp. 230-252.

2. Anderson, James P., ***Computer Security Technology Planning Study***, U.S. Air Force, ESD-TR-73-51, Vol. 2, October 1972.

3. Carlstedt, Jim; Bisbey II, Richard; Popek, Gerald, ***Pattern-Directed Protection Evaluation***, USC Information Sciences Institute, ISI/RR-75-31, June 1975.

4. Bisbey II, Richard; Popek, Gerald; Carlstedt, Jim, "Inconsistency of a Single Data Value Over Time," USC Information Sciences Institute, 1975.

INFORMATION AUTOMATION

Research Staff:	*Donald R. Oestreicher*
	Robert H. Stotz
	John F. Heafner
	Richard C. Mandell
	Jeff Rothenberg
	Ron Tugender
Research Assistant:	*Larry Miller*
Support Staff:	*Katie Patterson*

INTRODUCTION

Military command and control technologists are faced with a tremendous challenge. With the increasing sophistication of weapons systems and decreasing time frame for making decisions, it is essential to provide the military commander better quality information faster, even though manpower has been reduced by the conversion to all-volunteer forces. With today's technology, messages can traverse several thousand miles in fractions of a second, but hours are lost at either end, both in entering the message into the communications system and in delivering it to the man who can act on it.

The IA project is studying the application of on-line, interactive computer technology to the military message handling problem and is preparing an operational test. On the basis of the ARPANET message system experience, we are confident that such a service has a high payoff to the military. Not only can formal message preparation and delivery become faster and more reliable, but the processing facilities provided can also be put to new use. For example, with such a service the status of a message is automatically available at all stages from preparation to delivery. Much more detailed accounting and auditing is easy to maintain, providing a better understanding of the basic communication process. Entirely new facilities become available as well: for example, using the message service to alert individual users when certain events have occurred (e.g. "the message from Capt. Jones that you were expecting has arrived"). Automated suspense files, calendars, etc., are also simple to provide.

Perhaps most important contribution of such a system is that it makes available a secure, informal (off-the-record) message facility. This electronic memo pad is swift and convenient to use and, unlike the telephone, does not require simultaneous attention of sender and receiver.

The project is specifically directed to the military communication environment, and even more specifically to nonexpert users. The most effective way to introduce such a service into the military community is by means of an operational test at a military site, which will serve a twofold purpose: it will demonstrate the utility of an on-line message service in an environment credible and comprehensible to military planners, and allow system builders to understand the impact of such a system on the user organization and to evaluate the cost versus benefits of its various features.

BACKGROUND

Although the IA project actually began in the fall of 1973, its roots reach back to a three-week study, conducted on behalf of ARPA, of the military communications on the Island of Oahu [1]. This study was initiated at the request of the Secretary of Defense for Telecommunications as a part of a Navy program called COTCO, whose mission was to consolidate and improve communications on Oahu. Until ARPA'S involvement, COTCO advocated conventional data processing solutions. The ISI report, which recommended a complete island-wide interactive writer-to-reader message service electrically coupled to AUTODIN (the military's backbone communication system), was submitted by ARPA to DoD, where it excited considerable interest but was generally regarded as too radical to be included in a production system without a better appreciation of its cost and benefits.

INFORMATION AUTOMATION PROJECT

Against this background the IA project was started at ISI in the fall of 1973 with a twofold goal: 1) to develop the technology for providing on-line computer services directly to users who are neither specialists in computer science nor specifically trained operators and 2) to develop an on-line, interactive, writer-to-reader message service for the military community. The two goals are in fact indivisible. The military action officers who send and receive messages are not computer specialists. For the service to be useful, an interface must be provided that knows a great deal about each individual's habits, thus making his use of the service seem easy and natural to him.

Military Message Service

In the military, formal messages are archived for posterity, along with pertinent signoff data. Reference 2 describes in some detail how formal messages are handled today and how they are to be handled by the proposed IA message service. The concept is to put action officers directly on-line to a message service that provides interactive assistance for formal messages from the initial draft preparation through review and rewriting (the process termed coordination), through transmission and distribution to eventual recipients and finally to archival storage. In addition, the IA message service will provide informal "off-the-record" communication between users, a

service now unavailable (except by telephone or personal contact) but considered very valuable in accomplishing daily tasks.

Such an on-line message service, new to the military, allows coordination on draft messages without requiring face-to-face meetings, and permits rapid and secure formal or informal written communications. The anticipated benefits of this service are as follows: easier and faster message preparation and delivery, improved efficiency of action officers' time, better information dissemination, better understanding of information flow, and reduction of clerical load.

While these benefits all seem worthwhile, there is skepticism about whether they would in fact be realized. Before military decisionmakers are willing to invest in such a new facility, they would like to know the service's real value in an operational situation. As mentioned in the Introduction, the mechanism planned to determine this value is a formal test of the service in a working military environment, from which we hope to learn what features are valuable, how the service is used, and how it affects the way the user organization does its business. This information is essential for long-range military communications planning and for proper implementation of production systems.

The need to conduct this test has focused the IA project on designing a service that can be put into an actual military user environment. This focus requires a great deal of attention to functional performance, user interface, reliability, security, and scalability. This emphasis is required because current ARPA research products have not sufficiently addressed these issues for operational military environments. Also, to maximize the test results, the IA project has paid much attention to flexibility in introducing the service to users and in instrumentation for obtaining meaningful data about the effects of the system on users.

Functional Performance

To be a success, an on-line message service must provide the improvements inherent in automation without overly disrupting the traditional patterns and procedures that, though not ideal, are known to work. The manual nature of today's message service is somewhat cumbersome, but it is extremely flexible; each command or organization is able to tailor its procedures to its own needs. One of the unique characteristics of the IA message service is that it provides this tailorability.

To adequately support military message handling the organizational structure of the user community must be reflected in this service. For example, the rules about who can access what message files and who can release what messages must be carefully modelled. By definition, formal military message traffic flows between commanders of organizations, even though the messages nearly always originate and terminate at much lower levels. This necessitates special "coordination" or "staffing" procedures on outgoing messages (which require approval up the entire chain of command) and complicates the distribution of incoming messages. The IA military message service is unique in its approach to these problems.

It is also necessary that the on-line service be easy to use. It is certainly easier to type "send for coordination" than to hand-carry a draft message around to each coordinator. However, by automating this transmission we are faced with making the use of terminals competitive with paper and pencil. Toward this end the IA project is developing scanning and editing aids that currently do not exist. For instance, to facilitate integration of comments and changes from several coordinators, the service offers the ability to compare two versions of the same paragraph on separate windows of the CRT screen, highlighting the differences by making the changed characters brighter.

The proposed IA message service is divided into two stages: preparation and delivery. The former stage includes the creation of the draft message and the coordination of this draft with other users until it is signed off for release. For this stage the IA message service provides a special-purpose editing program which understands message formats and checks that the contents of the various fields are legitimate. The editor is structured so that a coordinator's editing of a message is stored as a special change file rather than as actual modifications to the original.

The author of the draft message controls the sequence and timing of delivery of the draft to coordinators. The message can proceed serially or in parallel (or any combination of the two). The author can have the message returned to him after each signoff (so he can incorporate the changes), he can ask that he simply be notified after each signoff, or he can let the coordination delivery proceed automatically.

Often a coordinator of a message wishes to obtain the opinions of others on his staff before he signs off. The IA message service allows the coordinator to "delegate" to as many people as he wishes the capability to comment and edit the message (each delegate edits from the original and creates his own change file). If so inclined, the coordinator may also delegate the signoff responsibility, but this is restricted to a single delegate only. The message service may retain all of this delegation information for audit purposes. During Phase 2 of the IA development (see the Project Plan subsection) this delegation facility will be extended to permit a user to specify in advance the criteria for selecting messages to be delegated to others. The service will then automatically perform the delegation whenever a message meeting these criteria is received.

This coordination process can be iterated as often as necessary, with each version being coordinated independently. A major research goal of IA is to learn more about the coordination process and about how to structure the computer-aided environment to enhance the effectiveness of this coordination.

The delivery stage involves conveying the message to its ultimate recipients, archiving it, plus providing aids for the user to sort his messages, scan them, and file them for later retrieval. The first step in this process is to determine distribution for the message. Because of the military policy that all formal traffic flows between commanders of organizations, it is necessary to employ complex procedures to determine the real ultimate recipients. The IA message service extends the normal

"one-pass" distribution algorithms provided in current AUTODIN terminals (e.g., LDMX) to allow each user to add his own personal distribution determination. A special form of distribution determination provided by IA, called Guarding, allows a user to specify criteria for messages that are to be routed to the first "on-line" user on the guard list. This assures that incoming messages meeting these criteria will be delivered to a live person who can act on it immediately.

A different form of special handling offered by IA is the alerting mechanism, which allows users to specify criteria for messages that will cause immediate action on the user's screen when they are received. This will notify the user of the event immediately, if he is on-line, or as soon as he comes on, if the event occurred while he was off-line.

Message selector criteria can also be applied to incoming messages to sort them into "folders" for the user. This provides the electronic analog of file cabinets. Since the message service can retrieve messages rapidly, these users' folders actually store only citations to messages, rather than the messages themselves, which limits the computer storage required to easily manageable size.

User Support

The ARPANET experience provides ample evidence that computer scientists can use on-line systems effectively with little or no formal training in their operation. There are also many examples of systems used every day by nonspecialists who have had intensive training (e.g., airline reservation clerks). To be effective, however, a military message service must be usable by non-computer people (action officers) with minimal formal training. Few officers spend more than 10 percent of their time in message-related functions; moreover, the present effort requires no specialized training. No on-line message service will be used in the military if it is not virtually self-evident and highly supportive whenever the user has any questions or difficulty. The IA project is focusing on this problem as a central research issue.

The approach chosen to provide the necessary support for the user who is not a trained operator or a computer specialist is to interface him to the message service through an "intelligent front-end process" which we call his "Agent." This Agent makes the service appear consistent to the user. It is designed to handle all control procedures (e.g., editing, help, defaulting, error handling, context mechanisms, etc.) in the same place and therefore in the same way throughout all phases of the service. This is a major source of difficulty in the current TENEX message facilities. The Agent and its components are described in detail in Refs. 3, 4, and 5. Briefly, it consists of a Command Language Processor, a User Monitor (with attendant background analysis processes), and a Tutor.

Command Language Processor (CLP). This serves as the interpreter for user commands operating from a dynamic input string and provides input editing functions and screen control. To support the neophyte, the CLP has a strong emphasis on error detection, recovery, and correction. It also acts as the

driver for the rest of the Agent, calling in the User Monitor and the Tutor when appropriate. The CLP operation is affected by User Profile data which provides information unique to each user.

User Monitor (UM) and Analysis Packages. The User Monitor collects data on user performance and provides the User Profile data used by other parts of the Agent (Tutor and CLP). Analysis programs process user performance data to test hypotheses and modify the User Profile.

Tutor. This provides intelligent help to on-line users by explaining commands, reporting errors, introducing new features, and providing reference documentation. Tutor operation is also affected by the contents of the User Profile.

The Agent is designed to collect specific data about the user's use of the service, to make certain analyses of that data and, on the basis of the results, to recommend changes to the way the user deals with the service or the way the service looks to him. After we have gained actual user experience, we fully expect to have to change the nature of the data collected, the way it is structured, and how it is analyzed. In this process, however, we expect to learn a great deal about the critical parameters of a man-machine interface and how to control them to maximize the user's performance and satisfaction.

As an initial effort in this area a pretest of three message service language forms--keyword, positional, and English-like--has been prepared. The goal of this pretest is to learn user preferences in language form in order to "normalize" the languages used (i.e., put each language on an equal footing with regard to the users and the tasks) for a later comparative test of user performance. For example, if the keyword language form required the user to type some long and irrelevant keyword each time a particular operator were needed, the user would be unduly prejudiced against the keyword language form. The Agent is designed to support multiple command language forms and individual variations of them. This facility will be employed in conducting the comparative tests and will be available for use in the operational tests thereafter.

Reliability

The IA project plans to make its message service reliable by making it a distributed process across multiple host processors and by keeping redundant copies of the service's basic files dispersed among these hosts. If any one host is down, any user can then still be served. Since the processes are distributed, a user does not need to run on the machine which stores his files.

In order to make this work, file naming conventions must be coordinated to insure system-wide uniqueness. In the proposed IA message service design there are three distributed processes, each of which controls a separate data base. The Coordination

Daemon controls all messages in preparation; the Transmission Daemon controls all messages that have been released; and the User Daemon controls all user personal data files Every host involved in the service has a copy of each of these daemons. When a user logs on, he is assigned to a host by the User Daemon. That host's daemons retrieve his personal files and then start up a job for him. This user job talks to the daemons for all its subsequent message file accesses. This distributed nature of the IA message service with redundant file storage provides the robustness required for a military environment.

Security

Another important requirement for this service is that it must meet military security specifications. While there are some systems-level issues not addressed by this project, the service is being designed to a consistent model of the necessary access controls to satisfy this need. Verification that the program actually matches the security model will be performed only at the top level.

Privacy (control of message access on criteria other than security level) is another major concern in a message service. The principal difficulty here is in eliciting from the military a reasonable statement of what the rules should be. "Need to know" is a highly judgmental quality and very difficult to model. The IA project plans to embody access control mechanisms general enough to be applied to a broad set of models. When a particular privacy model is elaborated, it should be easy to implement. Initially, the service will support author-assigned access control at the message level.

Scalability

In the COTCO study it was learned that on the average day on Oahu, 6,000 formal AUTODIN messages are sent out and 15,000 messages are received. To insure that received messages get to the appropriate people, an average of 40 copies are distributed. The CINCPAC communication center devotes a 24-hour-a-day printing press to this function. To handle traffic of this magnitude in an on-line system, it is necessary to organize the messages as efficiently as possible. For example, when a message is "delivered," instead of making a private copy for each recipient (as is done with current ARPA message services), the IA system delivers a brief "citation" to the message. The message itself is stored in two central locations (redundancy for reliability). The user is then granted read-only access to one of these central copies when he wishes to read its contents.

Other design decisions in the IA message service also reflect this concern for scalability. The organization of user files is also done by a central process (User Daemon), to compact them as much as possible. In this way, data relevant to many users can be kept in the same TENEX directory rather than requiring a directory per user. The daemons are distributed processes that operate across multiple hosts on the network so that the service can grow in a straightforward way by expanding the subnet (more nodes and more links) and adding more message processors.

Test Objectives

Perhaps the most important aspect that distinguishes the IA project from previous message service developments is that it is being designed from the ground up to be used by the military in a test situation. The data collection and analysis being done by the Agent to measure the users' activity on the service is highly relevant to the test objectives of the military (i.e., to understand the impact and utilization of the service). Additional functional information (such as traffic, patterns, attributes of messages, etc.) will be collected by the daemon processes as appropriate.

PROGRESS TO DATE

A year ago there were two projects at ISI that were relevant to the current IA project. Command and Control Message Processing Technology (CCMPT) was identifying research problems and opportunities for applications for interactive message processing services in the military environment. Information Automation (IA) was studying the architecture for on-line message services for computer-naive military users. In September 1974, six reports (Refs. 2-7) were published which explained the IA basic design approach and some of the underlying philosophy. Reference 2 describes the functional performance for an on-line military message service. References 3,4,5, and 7 describe the user support environment for such a system, including the Command Language Processor (CLP), the Tutor, the Editor and a methodology for refining command languages. Reference 6 covers executive system support required. Reflected in these documents is the information gathered from many discussions with military communications specialists at installations such as CINCPAC, NAVELEX, NAVCOSSACT, Naval Research Laboratory, Naval Electronics Laboratory Command, MITRE, Army Communications Command, Army Materiel Command, Air Force Logistics Command, Air Force Communications Systems Command, and others. These reports have been useful in communication with the ARPA research community about the basic problem area and have served as guides to several ARPA contractors. They have also been a source of feedback from the military community.

As the plans for conducting an operational test of military message processing on Oahu began to take shape last fall, CCMPT and IA were merged into a single integrated program focused on this test. An informal design report was produced for ARPA review in January. This report describes the detailed design of the IA message service, including such information as message representation and data formats. The design called for implementation of the message service on a standard TENEX using a modern CRT terminal. It was planned that the message access mechanism would be based on the NLS routines being produced by Stanford Research Institute for use as a tool on NSW.

Upon approval of the design in March 1975, work began on implementation of this message service. Since that time significant portions of the CLP and terminal interface

have been completed. Delays in production of the NLS routines for NSW forced a reassessment of their utility to the IA program and in early May plans to use NLS were dropped.

In the area of the user interface, the project has developed a command language protocol analysis test to be applied to eventual users of the test message service as described earlier. The purpose of the analysis is to provide the command language designer the most representative language of each of several language forms tested. This command language protocol analysis test has been conducted at ISI using ISI staff as subjects, in order to evaluate the utility of the approach; Ref. 8 reports the test findings. An improved version of the test now being developed will be used in testing the ultimate military users.

In addition to designing the message service, the project has worked with ARPA, NAVELEX, and CINCPAC to develop tentative plans for operational testing of the interactive message service and for transferring the technology involved into the hands of those people within the Navy who can use it to implement future systems. The test plan involves running approximately 25 terminals connected to a message service on a dedicated TENEX computer on the ARPANET. This test will be run in a system-high security mode; that is, terminals will be restricted to a large controlled-access area (blockhouse) and all users will be cleared to the highest classification of traffic handled. The message service provided will have a connection to the LDMX at Camp Smith to provide operational traffic. Efforts are under way to get active participation of NAVCOSSACT personnel. The intent is to foster the technology transfer required for optimum impact on the defense community. Effort is continuing to structure a test plan to insure that major questions about the inpact and utility of interactive services will be answered definitively.

PROJECT PLAN

The steps ahead for the IA project are to complete the implementation of the design, test it on friendly users (making appropriate improvements), install it in an operational environment, and conduct operational tests. This program requires several phases of activity, as enumerated below.

Phase 1

Phase 1 will implement the Agent and the creation and coordination aspects of the IA message service, providing a highly interactive, useful service on which to initially test the concepts underlying the Agent and gain some valuable feedback from military users on the system's functional performance. This first phase will be implemented first on a single processor and thus will not test the mechanisms for distributed processing and backup files.

This coordination service is intended to provide, as output, either standard ARPANET mail service messages or properly formatted input to AUTODIN. Thus this service will be useful in a controlled military environment or as a subsystem on TENEX on the ARPANET. Since the Phase 1 product is a subsystem relying on other TENEX subsystems for message delivery, reading, and archiving, this message service will not yet have the single, consistent, homogeneous look to its users that is necessary for success. However, Phase 1 will serve as a basis for an initial evaluation of this type of service for the military. This service will be ready for initial testing in the first quarter of calendar 1976.

Phase 2

The second phase of development will add to the service the processes associated with delivery, reception, archiving, and retrieving of traffic. In addition, the system will be extended to provide backup files and fully distributed daemon processes.

In Phase 2 the Agent development includes implementation of tutorials, more complete tutor data bases, and more powerful CLP screen control. In addition, it covers tailoring the service to the test environment and conducting user tests on the service provided by Phase 1. The Functional Module will be extended to handle the formal message reception features, providing automatic folder processing, delegation, and alerts. The daemons will be made distributed processes with backup files. Phase 2 is scheduled for completion by the fourth quarter of 1976.

Subsequent Phases

Phases have been identified for debugging and tuning and for the test itself. Details of these phases will be established through coordination with the many parties involved.

REFERENCES

1. Ellis, T. O., Gallenson, L., Heafner, J. F., Melvin, J. T., ***A Plan for Consolidation and Automation of Military Telecommunications on Oahu***, ISI/RR-73-12, May 1973.

2. Tugender, R., and D. R. Oestreicher, ***Basic Functional Capabilities for a Military Message Processing Service***, ISI/RR-74-23, May 1975.

3. Rothenberg, J. G., ***An Intelligent Tutor: On-line Documentation and Help for a Military Message Service***, ISI/RR-74-26, May 1975.

4. Heafner, J. F., ***A Methodology for Selecting and Refining Man-Computer Languages to Improve Users' Performance***, ISI/RR-74-21, September 1974.

5. Abbott, R. J., ***A Command Language Processor for Flexible Interface Design***, ISI/RR-74-24, September 1974.

6. Mandell, R. L., ***An Executive Design to Support Military Message Processing Under TENEX***, ISI/RR-74-25 (in progress).

7. Rothenberg, J. G., ***An Editor to Support Military Message Processing Personnel***, ISI/RR-74-27, June 1975.

8. Heafner, J. F., ***Protocol Analysis of Man-Computer Languages: Design and Preliminary Findings***, ISI/RR-75-34, July 1975.

NETWORK SECURE COMMUNICATION

Research Staff:	***Danny Cohen***
	Thomas L. Boynton
	Stephen L. Casner
	E. Randolph Cole
	James Koda
	Robert Parker
	Paul Raveling
	Dono Van-Mierop
Research Assistant:	***John K. Kastner***
Support Staff:	***Nancy Dechter***

INTRODUCTION

The major objective of ARPA's Network Secure Communication (NSC) project is to develop and demonstrate the feasibility of secure, high-quality, low-bandwidth, real-tim , full-duplex (two-way) digital voice communications over packet-switched computer communication networks. This kind of communication is a very high priority military goal for all levels of command and control activities. In 1972 the House Special Subcommittee on Defense Communications reported that the most prominent equipment deficiency experienced in Vietnam was the lack of ability to encrypt voice transmissions. ARPA's NSC project will supply digitized speech which can be secured by existing encryption devices.

The major goal of this research is to demonstrate a digital high-quality, low-bandwidth, secure voice handling capability as part of the general military requirement for worldwide secure voice communication. The project goals are to be achieved within the context of operational military requirements. However, it is expected that early use by the military will be on an experimental basis, t provide an opportunity to add system improvements unique to the military.

ISI's role in ARPA's project is as follows:

- To continue developing the Network Voice Protocol required for communication of coded speech over a packet-switched network in real time.
- To develop on-line voice conferencing capabilities.

- To continue implementation of the PDP-11/SPS-41 system for real-time LPC vocoding.
- To develop dynamic off-line voice systems for storage and retrieval of voice files.
- To integrate a signals and voice input system, using the Voice Recognition techniques to be developed by Lincoln Laboratory.
- To integrate an authentication and privacy mechanism to be developed by Speech Communications Research Laboratory (SCRL).

RESEARCH APPROACH

Several different problems must be solved in order to advance the overall state of the art of real-time speech communication. These problems range from acoustic research into vocoding techniques to networking research for achieving the required performance.

There is only one criterion for judging such systems, namely, how useful they are for their users. Therefore, all the related efforts always begin with the user. First, the user interface (procedures) were designed, then the system protocols necessary to support the user interface were designed, then the system to support these protocols were designed and implemented. This outside-in approach proves itself time and again for most applications with a man-in-the-loop.

There is a definite correlation between compression and computation. The higher the compression, the more computation required. Similarly, there is a connection between quality and compression. The better the quality, the higher the bandwidth required (assuming the same computation). There is also a three-way relation between the network-related parameters: bandwidth, delay, and continuity.

One of the purposes of the research is to optimize simultaneously the speech compression and the network performance required for real-time communication.

ISI conducted network-related research and built real-time systems, integrating algorithms developed by other ARPA sites, such as Lincoln Laboratory, Speech Communication Research Laboratory, Stanford Research Institute (SRI), and Bolt, Beranek & Newman (BBN).

PROJECT GOALS

A prototype of a PDP-11/SPS-41 digital voice communications system was implemented at ISI. The Network Secure Communication project at ISI has developed this system in order to demonstrate the feasibility of secure, high-quality,

low-bandwidth digital voice communication in real time over a packet-switched computer communications network.

Another objective of the NSC project has been to provide a framework within which to do digital voice conferencing and to demonstrate a working conferencing system. This is the logical extension of simple person-to-person voice communication; a conferencing capability would greatly extend the usefulness of a secure digital voice communication system.

CURRENT STATUS AND ACCOMPLISHMENTS

The Network Voice Protocol (NVP)

ISI developed the NVP early in 1974. In August 1974 it was successfully used for high rate real-time communication between ISI and Lincoln Laboratory using CVSD, and in December a more expanded version of NVP was successfully used for lower rate LPC communication between Lincoln Laboratory and Culler-Harrison Inc.

NVP has some unique features which make it basically different from other existing Host-to-Host protocols on the ARPANET.

NVP takes advantage of the properties of human speech, such as silence periods, in a way which optimizes the communication by reducing the required bandwidth.

NVP is geared for operation in any packet-switched network rather than specifically designed for the ARPANET.

NVP separates control from data in a way which allows interfacing of encryption devices for data only, without affecting control data which cannot be end-to-end encrypted. Control issues like timing and order of arrival are separated in a way that allows future network protocols (at the HOST/IMP level, like the Kahn-Cerf protocol) to handle them. NVP can use both Type 0 messages (fully controlled, guaranteed, synchronized, and sorted by the SUBNET) and Type 3 messages (which are not).

NVP is designed to trade reliability, if needed, for higher bandwidth and lower delays. Its operation never depends on the arrival of all the messages.

NVP is designed to separate vocoding-dependent issues from the vocoding-independent ones. This allows easy incorporation of new vocoding techniques as they become available. At present there are NVP interfaces for LPC and CVSD only. NVP includes its own initial-connection procedure, which is different from, but similar to, the standard "ICP." Its main objective is ***negotiation***, a stage needed for defining the format to be used later for data-transfer.

NVP allows systems with different levels of NVP implementation to be compatible if the set of their mutually implemented features is found to be sufficient.

NVP allows any message (control or data) to be lost or delayed without catastrophic effect on the communication.

NVP is designed to eliminate the possibility that one system can tie up the resources of another system unnecessarily.

A full description (down to the bit level) of NVP will be found in an ISI research report (ISI/RR 75-39) to be published soon.

Linear Prediction Coding (LPC)

Virtually all of the work done by the NSC project has been along critical paths leading to a low-rate, high-quality, real-time LPC vocoder for digital speech transmission.

The accomplishments most directly related to LPC are:

1. Creation of an efficient environment in which to create and run highly complex, dynamic software systems (such as LPC) on the SPS-41.

2. Systems design of the LPC software for both the SPS-41 and the PDP-11, and implementation of the SPS-41 LPC analysis and the entire PDP-11 LPC system.

3. Overcoming the difficulties which arose from the low reliability of the SPS hardware, and helping the SPS Corporation in debugging their hardware. This led to our ability to run the SPS system in a relatively reliable fashion.

The software environment for LPC. Very little software was delivered for the SPS-41. What software existed was designed for applications in which a single program was loaded into the machine and run indefinitely. LPC required the ability to write many SPS program modules, test them separately, and then integrate them into a system in which program modules are loaded into the SPS, run to completion, and dynamically overlaid with the next module needed. The final LPC system requires many such modules.

The following software was written or modified for this purpose:

- The SPS-41 assembler (BOXASM) was modified extensively. BOXASM is written in FORTRAN and runs on the PDP-10.

- An Automatic Reformatter (ARF) was specified by ISI and written by ISI and BBN. The purpose of ARF is to transform the output of BOXASM, an ASCII character file, into another ASCII character file which is then input to a PDP-11 assembler or cross-assembler. Thus an SPS program is

transformed into data blocks which can be loaded into PDP-11 memory for dynamic loading and execution. ARF is written in FORTRAN for the PDP-10.

- The PDP-11 cross-assembler MACN11, which runs on the PDP-10, was modified extensively for this and other purposes.

- The operating system ELF provided network access for the PDP-11 support program for LPC. Because ELF was in an "almost finished" state, extensive work was required to make it functional for our application. In particular, considerable time was involved in bringing the virtual memory capabilities of ELF to operational status.

- An executive program (EXEC) was written to permit dynamic loading and execution of SPS program modules. The EXEC is an SPS-41 program which resides permanently in the SPS and controls its operation. Commands for the EXEC are stored in PDP-11 memory locations, along with SPS-41 program modules which the EXEC loads into the SPS and starts. The EXEC has facilities to detect when each SPS program module is done, and can even load the next program module while the present module is running.

- FT11/FT10 is a user/server pair of programs which run in the PDP-11 and PDP-10, respectively, to transfer files between the two machines using a 2400 baud line. File transfer is required because source files are maintained and assembled on the PDP-10.

The SPS-41/PDP-11 LPC system. The SPS-41 LPC system consists of an analyzer written by ISI and a synthesizer written by SRI. The analyzer consists of a series of seven SPS-41 program modules, and the synthesizer consists of one large program module. The SPS-41 LPC analysis is made up of four basic modules, some of which are used more than once in each analysis frame. They are as follows:

- A windowing, autocorrelation, and normalization module, used once during coefficient analysis and twice during SIFT pitch extraction

- A matrix inversion module, written at BBN, used once during coefficient analysis and once during SIFT pitch extraction.

- A low-pass filter and downsample module for SIFT pitch extraction.

- An inverse filter (convolution) module for SIFT pitch extraction.

Block diagrams of the SPS-41 analysis and synthesis systems are shown in Figures 6.1 and 6.2.

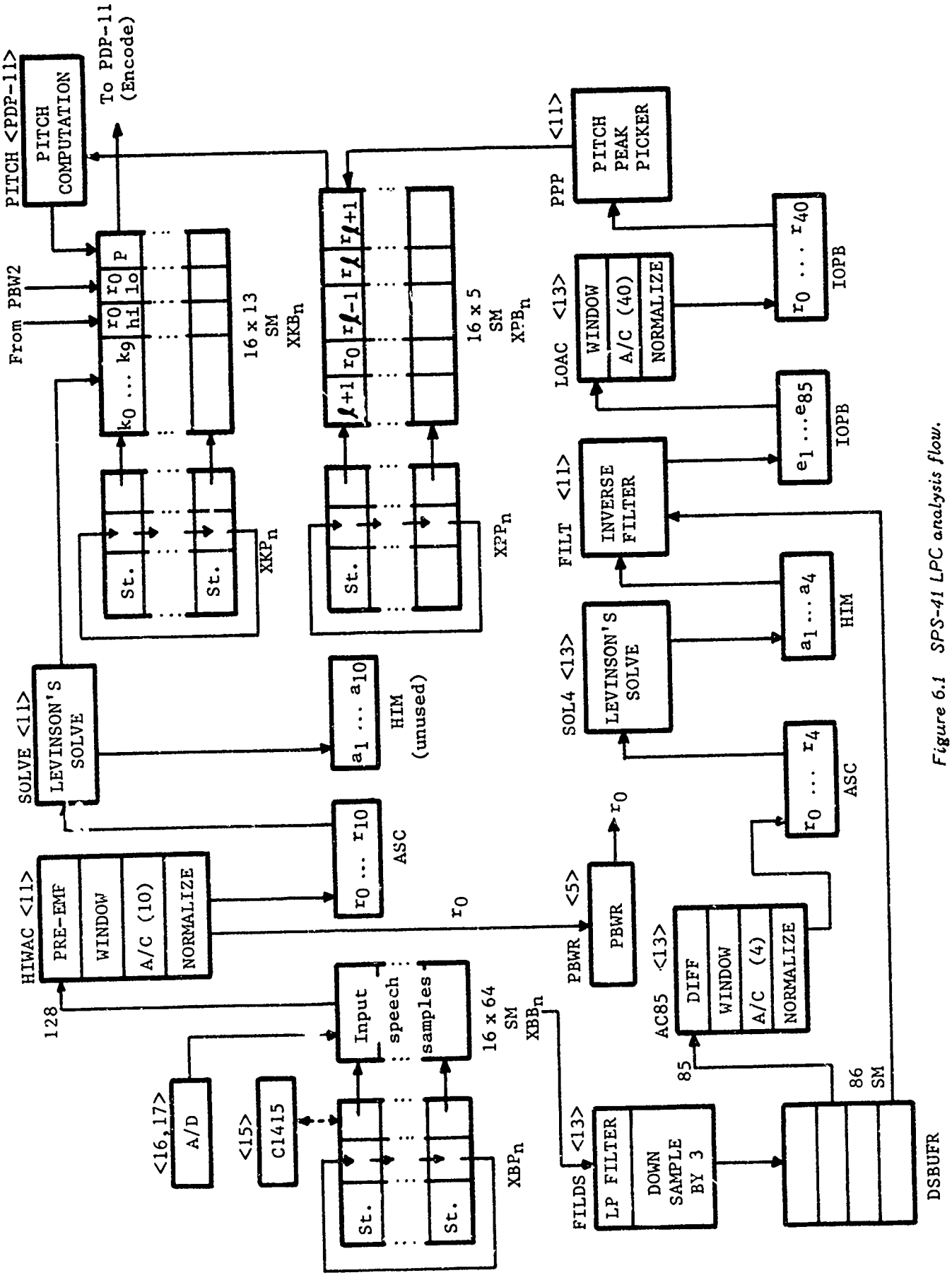

Figure 6.1 SPS-41 LPC analysis flow.

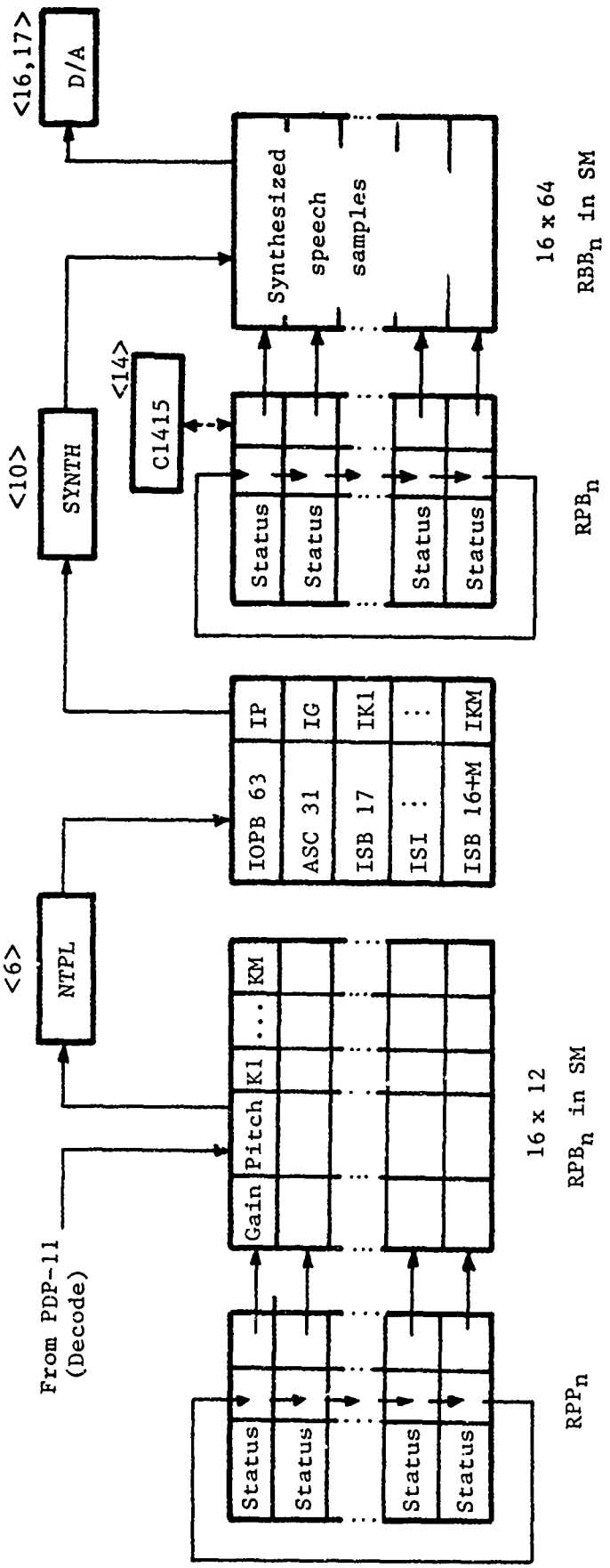

Figure 6.2 SPS-41 LPC synthesis flow.

The PDP-11 LPC system has the following components (not including the ISI-modified ELF operating system):

- A controller module, which supervises the PDP-11 LPC process and handles communications to and from the ARPANET via the NVP.
- A transmitter module, which takes parcels of compressed speech data from the SPS-41 LPC analyzer, encodes them (in the information-theory sense), and formats them for transmission as ARPANET messages. The encoding is done by a special subroutine which can be changed to accommodate various types of vocoders.
- A receiver module, which takes incoming speech data from the ARPANET, decodes them, and passes them to the SPS-41 LPC synthesizer. The separate decoding subroutine can also be changed to allow vocoders other than LPC to be used. This module was written at SRI.

It is important to note that the PDP-11 LPC system is quite flexible, and not limited only to LPC; only the encoder and decoder subroutines are specific to LPC. This allows other vocoding methods to be implemented with relative simplicity.

Continuously-Variable Slope Delta Modulation (CVSD)

Continuously-Variable Slope Delta Modulation (CVSD) is a speech-oriented bandwidth compression technique which compresses speech to within the range of 10 to 20 Kbps, with less quality than LPC. However, the computation required for CVSD is only a small fraction of that required for LPC. Therefore, CVSD can be easily performed by either software or hardware.

Procurement bids for hardware CVSD vocoders were issued in 1974 according to specifications issued by Lincoln Laboratory, and several units were purchased from General Atronics. The CVSD hardware devices will play a major role in the initial conferencing experiments and in other systems which require more than one vocoder per site, since the cost and complexity of any LPC implementation makes the use of more than one LPC vocoder per site prohibitive.

CVSD software network communications experiment. In April 1974, ISI had implemented an off-line simulation of CVSD on the SPS-41. By June, ISI had designed and implemented a large and complex CVSD program for the SPS-41 which would allow on-line communication over the ARPANET. This program was formerly one of the most complex ever run on the SPS-41, using eight of the machine's sixteen input-output processor channels.

A PDP-11 system to handle CVSD communications between the SPS-41 and the ARPANET was designed and implemented in parallel with the SPS CVSD effort. This PDP-11 system was designed to run under the ELF operating system for the PDP-11. In early August (within three weeks after the necessary ELF facilities were completed

by SCRL) the on-line CVSD system was brought up and experiments in digital speech communication with Lincoln Laboratory were begun between ISI's SPS-41/PDP-11 system and Lincoln's FDP/TX-2 system at a data rate of 10,000 bps. The first Network Voice Protocol (NVP) was used for these experiments. This was the first use of a packet-switched network for digital voice communication. The experiments were completed in October 1974.

CVSD hardware. In January 1975 five of the hardware CVSD devices built by General Atronics from specifications by Lincoln Laboratory were delivered to ISI. These devices are capable of CVSD at data rates from 8 Kbps to 18 Kbps. It was immediately apparent that although the devices themselves performed well, the interface provided to connect the devices to the PDP-11 would cause an impossible load on the PDP-11. This initiated the development of an improved interface, the PB11-A. The PB11-A allows serial communication between the PDP-11 and the CVSD boxes, over long cables, making it possible to use them away from the computer room. It combines the data to (and from) all the boxes so that the computer is interrupted only once every 16-bit period for all boxes rather than once for each bit from each box. The PB11-A also solves a synchronization problem by providing a single clock signal for all the CVSD boxes.

Support Software

A very large portion of the work required for a large system such as LPC or CVSD is spent in writing support software of all types. The following is a list of the major items of support software written by ISI:

PDP-10 programs.

- ARF: The Automatic Reformatter for SPS-41 programs, as described in the LPC section.
- FT10: The PDP-10 side of a file transfer program which transmits files from PDP-10 disk to PDP-11 disk and vice versa.
- 11COPY: The predecessor of FT10/FT11. Transfers load modules to the PDP-11.
- MACGEN: A text generator for easy creation and formatting of PDP-11 assembly language (MACRO-11) programs.
- DUMP11; An ELF core dump formatter.

SPS programs.

- EXEC: The SPS executive described in the LPC section.
- ASFLG: A test program to test the SPS AS flag, which did not work as described in the documentation.

- ECNS: A set of 6 programs to test SPS ECNs. Supplied to all network SPS users.

- D2AX: A test program for the SPS D-to-A and A-to-D converters.

- DCVSD: A dual port CVSD program.

- LPC: The SPS implementation of linear predictive coding (LPC) itself. It includes the following program modules:

    - ADDA: The A-to-D and D-to-A handler.

    - FILDS: Low pass filter and downsample module.

    - HIWAC: Windowed double-precision autocorrelation module for coefficient analysis.

    - SOLVE: Robinson-Levinson recursive matrix solution module for coefficient analysis.

    - AC85: Windowed double-precision autocorrelation module for pitch analysis.

    - SOL4: Robinson-Levinson recursive matrix solution module for pitch analysis.

    - FILT: Inverse filter module for pitch analysis.

    - LOAC: A second windowed double-precision autocorrelation module for pitch analysis.

    - PPP: A pitch peak picker.

- XLPC: A test program for testing LPC under controlled conditions.

- CLPC: A second test program for testing LPC under controlled conditions.

DOS programs.

- LOAD12, LOADSP, LOADER, LOADNC, and LOADAL: Various loaders for the PDP-11.

- FILSTT: A PDP-11 file status package.

- FT11: The PDP-11 half of the file transfer package.

- EXECLD: Loads in the SPS overlay exec from disk (used as a command extension to SPUD).
- LDSPS: Sets the various channels of the SPS to a predefined set of addresses.
- STCHAN: Starts the SPS (using a particular sequence needed by the SPS).
- INITAL: Sets the various channels of the SPS using LDSPS to start an initialize program (INIT).
- INIT: Pulses a channel to set high address bits.
- ENCODE: Encodes LPC parcel information into a network message for transmission.
- SPTEST: Runs ENCODE and DECODE back-to-back so that local tests of LPC may be made without the ARPA network (used for debugging).
- LPCTBL: Is the table used by ENCODE and DECODE for compression.

ELF programs.

- IKINT: A module which allows the ELF user to field interrupts.
- IKUSBO: A program that allows ELF users to bootstrap load modules into user space.
- BEEBUG: A powerful debugger which runs in ELF user mode on a Beehive terminal.
- TCP: A TENEX compatibility package which provides I/O capabilities for ELF programs somewhat like those in TENEX.
- MASTER: A mini-executive to control CVSD running under ELF.

ELF modules (components of the ELF operating system) for ELF-II.

- KDIMP: The IMP driver module.
- KDPR11: The paper tape reader driver module.
- KDRK05: The RK05 disk driver module.
- KTMP: The programmable clock driver.
- KTML: The line clock driver.

- KTMI: The real-time clock driver.
- NIO: The ELF network I/O driver.

ELF modules for ELF-I.

- ELFNIO: The ELF-I network I/O driver.
- ELFCLK: The ELF-I driver for the PDP-11 clock.
- ELFTC: ISI modified to add simple FTP features.
- ELFRK: A disk driver for the simple ELF FTP features.

Software extensively modified by ISI.

- MACN11: The PDP-10 cross-assembler for the PDP-11. Several new features were implemented and all known bugs fixed.
- IMPTST: The PDP-11 IMP interface test exercise program.
- IMPDIA: The PDP-11 IMP interface diagnostic program.
- SPUD: The PDP-11 program which exercises single programs running in the SPS-41.

SPS Status

Hardware debugging. The SPS-41 was installed in late 1973. Programming began in April 1974, and a two-month period of hardware debugging followed in April and May. During that period numerous bad components and broken wires and one design error were found and replaced or corrected. The SPS-41 generally ran dependably until September.

In September, the ISI SPS-41 was retrofitted for dual-port memory. Dual-port memory was necessary for LPC in order to allow the PDP-11 to compute while the SPS-41 is accessing memory. After work was begun on LPC programs which use the dual-port memory extensively, a large number of hardware design problems were found in the area of the dual-port interface.

Throughout the process of debugging the SPS machines, ISI has served as the central clearinghouse for information about SPS hardware bugs. In addition, ISI people have spent large amounts of their time to isolate the hardware bugs and write test and diagnostic programs.

To date ISI has generated many SPS programs which serve as effective diagnostic and test programs, and made these programs available to other NSC group sites. SRI,

SCRL, BBN, LL, and SDC have used some of these programs as part of their acceptance tests.

SPS hardware status. At the moment it is not clear how many bugs, if any, remain in the SPS-41's dual-port interface. SPS conducted a design review during early 1975 and failed to uncover any major problem. However, field debugging mostly by the designer of the hardware revealed too long signal paths which led to a series of ECNs, ranging from replacing 74XX ICs by 74HXX and 74SXX ICs, to operation with memory slowed down to 800 nanosecords (instead of 600 nanoseconds). Up until the issue of this report, the SPS hardware still was not operating in a reliable mode.

SPECIAL PROJECTS

Research Staff: *Stephen D. Crocker*

Ronald L. Currier
Norton R. Greenfeld
Dono van-Mierop

INTRODUCTION

Since its inception, ISI has undertaken several hardware development efforts in support of research requirements or to demonstrate a capability for a recognized DoD application. As reported in Ref. I, one of the most significant of these projects is the development and use of the Xerox Graphics Printer (XGP), a high-quality document printing capability in the form of a network terminal.

Two XGP systems have been installed, one at ARPA and one at ISI. They provide high-quality on-line hard copy with proportional spacing of characters according to width, and use of multiple fonts. This report is an example of the XGP's capabilities.

The hardware components of the XGP systems at both ARPA and ISI consist of a modified Xerox machine interfaced to a PDP-11/40 with 32K words of core and 256K words of disk, interfaced via a 2400 baud line to the ARPA TIP, which is driven over the ARPANET by any TENEX system, particularly OFFICE-1, ISI, and ISIB. See Fig. 7.1.

The software components of the XGP system consist of the following:

- A number of character set descriptions which give the correspondence between character codes and arrays of points,
- XOFF, an elaboration of RUNOFF which accepts a text file, performs filling and format calculations, and creates a file with text and control codes,
- XLIST, a transmission program which accepts either normal text files or files produced from XOFF, and transmits them to the PDP-11 over the ARPANET,

- A PDP-11 program which receives text over the ARPANET, justifies lines, converts character codes to dot arrays, and drives the XGP.

The software in use was adapted from software written at Carnegie-Mellon University. The primary changes have been to use ARPANET communication facilities to replace CMU's hardwire connection between the PDP-10 and the PDP-11, and to use TENEX file name conventions.

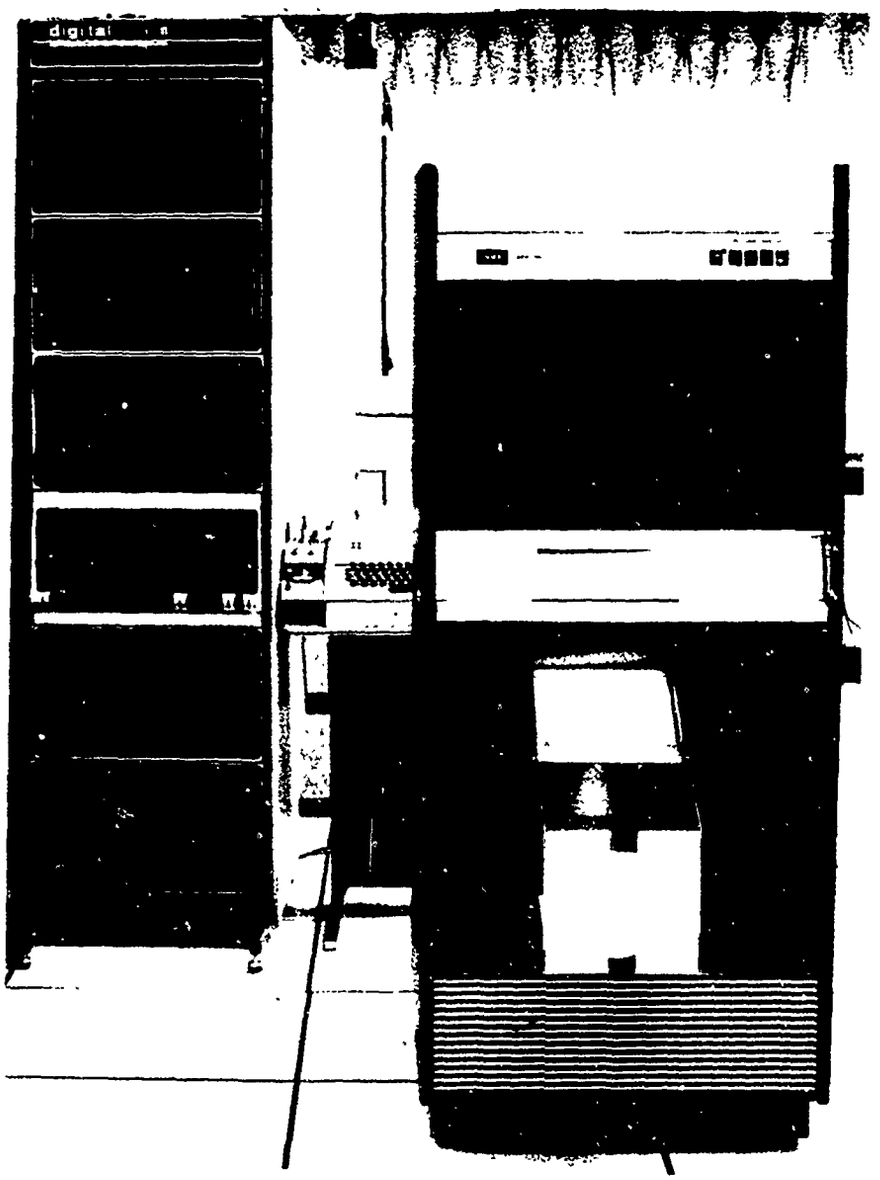

Photo by Marti Coale

Figure 7.1 Xerox Graphics Printer and its processor.

SYSTEM OPERATION

Operationally, the process of printing a file on the XGP has three distinct stages:

- Document preparation.
- XGP preparation.
- Text shipment and printing.

Document Preparation

Text is entered into the computer using any of several text editors. The user may include within the text various directives to control justification, filling, font changes, and so forth. If such directives are included, the user must run the program XOFF to convert his file to a form acceptable to XLIST.

XGP Preparation

The XLIST program is used to communicate with the PDP-11. The XGP has many parameters, such as page size, margins, and character sets. Currently, XOFF can insert commands in the document file to control any of these parameters except the shipping of character sets. Only some of the character sets are stored permanently on the PDP-11's disk. Character sets required during the printing process which are not on the disk must be transmitted to the PDP-11 before the document is transmitted. Transmission of character sets is usually initiated by issuing commands to XLIST.

Text Shipment and Printing

The XLIST function "*Perform Print*" sends a file to the PDP-11 and tells it to start printing. The program in the PDP-11 copies the document to its disk and then outputs it to the XGP, converting character codes to point matrices and expanding spaces to justify lines.

OPERATIONAL EXPERIENCE

The XGP systems have been used extensively since their installation. Although users are generally able to use these systems effectively, two major defects were noted: slow speed and difficulty of use.

Speed

Under the current design, files are copied completely over the ARPANET to the PDP-11 disk before printing is started. The printing process is governed by a paper speed of 0.67 inches per second, so that it takes 16.5 seconds to produce an 11-inch page. A page of printed output corresponds to about 2000 8-bit characters or 16,000 bits. Thus the printing process operates at about one kilobit-per-second. However, measurements of the current software show that transmission over the ARPANET operates in a range 200 to 800 bits-per-second, so that between 55 percent and 85 percent of the total time required to print a document is spent purely in its transmission over the network. By comparison, transmission from ISIB to the ISI XGP over a direct, non-network connection tends to operate in the 2000 to 5000 bit-per-second range, so that not more than 33% of the throughput time is due to transmission. Similarly, file transfers over the network which do not involve the XGP also operate well in excess of 2000 baud--usually closer to 7000 baud and sometimes at 25,000 baud.

A short investigation was conducted to find the location of the bottleneck. First, the physical connection between the TIP and the PDP-11 was increased from 2400 baud to 9600 baud, but no increase in speed resulted. Second, XLIST's communication strategy was changed. XLIST presently uses pseudo-teletypes connected to network connections, and it is known that this scheme is much slower, although more flexible, than using direct network connections. An experimental version of XLIST which used direct network connections was tested, and the throughput was raised to just under 1000 bits per second. Finally, the TIP buffer space was doubled and no increase in throughput resulted.

With the exception of the marginal increase in speed when using direct network connections, the low throughput does not seem correctable within the present hardware framework. In particular, the fact that regular file transfers between two hosts run an order of magnitude faster than our connection through the TIP strongly suggests that the TIP is incapable of supporting high throughput to a terminal connection. As a separate confirmation, the TIP is known to allocate only one message at a time on each of its connections, thus insuring long delays between messages and consequently low throughput.

Ease of Use

There are two major aspects of the current system design which make it much harder to use than necessary.

1. Users must command XLIST to send to the PDP-11 character sets which are required during the printing process, although they have already had to specify the same information once in the preparation of the text file.

2. Users must wait for the completion of the transmission of their files to the PDP-11; if they try some other action, the transmission is aborted. Since the user generally has no further need to interact with the XGP system once he has started the transmission of his document, it would be far better to transmit and print files as a background task.

CURRENT ACTIVITIES

On the basis of the accumulated experience and analysis of the problems, work was undertaken to modify the XGP systems to provide higher throughput and easier use. The steps being taken are as follows:

1. The connection between the PDP-11 and the TIP is being changed to use a host interface. Corresponding changes in the software in the PDP-11 will also be made.

2. Printing will be overlapped with transmission to achieve maximum throughput.

3. Shipment of character sets to the PDP-11 will be performed automatically.

4. The core allocation scheme is being revised to work with more than two fonts.

5. Defaults are being established so that the user only has to supply the name of the file to be printed.

6. A background process much like the LPT server is being developed.

Connection of the PDP-11 as a Host

In order to support high-speed network transmission, the hardware of the PDP-11 is being augmented with a host interface and enough memory to support both the ELF operating system and the XGP program. The total core on each PDP-11 will be 64K instead of the present 32K. Memory mapping hardware is also being added.

The current software, which is based on CMU's PDP-11 XGP program, will be replaced by a combination of VM ELF and MIT's XGP program. The VM ELF system will provide network and disk I/O and address space management. MIT's XGP software is a much improved version of CMU's software, providing the same functions of converting character codes to raster lines suitable for transmission to the XGP hardware.

Overlap of printing with transmission

Text received from the ARPANET will be buffered onto the disk. Printing will be initiated when text for a small number of complete pages has been received. If transmission is slowed after printing has started and the printing process actually catches up to the transmission, printing will be interrupted at the next page boundary. Printing will be resumed when recomputation of the throughput again shows it to be safe.

In normal cases, throughput of a few kilobits-per-second is all that is required to keep up with the printing process. Even when TENEX is heavily loaded, it should be able to accomplish this. Buffering is continuous across file boundaries, so printing should be continuous as long as there are files to print.

Automation of shipment of character sets

The background TENEX process will accept commands from the text file which ships character sets to the PDP-11. Corresponding changes to XOFF to generate commands have been made in part and will be finished. Character set names will be standardized, and the sets resident on the PDP-11 disk will be protected. Other character sets will be shipped automatically before each file is printed, and cleared afterwards.

Revision of core allocation

The present core allocation scheme in the PDP-11 prc .m places each new font in progressively increasing memory locations. Eventually, memory space is exhausted and the printing process is aborted. Since only two fonts are active at any one time, it is possible to reuse the space released by previously used fonts. A strategy to reuse the core space is being designed and implemented.

Establishment of defaults for XLIST and the PDP-11

Defaults for paper size, margins, character sets, and tab stops will be established so that line printer-type files will print as much as possible as they would on the printer.

Queueing of files

The functions of the current XLIST program will be divided into two parts. One part can interact with the user to accept filenames and destinations. It will copy the file into an XGP-PRINTER directory. The second part will be a set of permanent background tasks which will attempt to connect to the XGP's and will send files stored in the XGP-PRINTER directory to the designated XGP. There will be one background task for each destination XGP accessible to the host.

On the theory that anything queued for printing must eventually be printed, no priority or interrupt mechanism is being designed. Some thought will be given to this as we progress, however, since connecting the PDP-11 as a host would permit centralized queueing control and status reporting.

SCHEDULE

It is expected that these changes will be complete and the new system operational in the fall of 1975.

REFERENCE

1. ***Annual Technical Report, May 1973 - May 1974***, USC/Information Sciences Institute, ISI/SR-74-2, 1974.

ARPANET TENEX SERVICE

System Staff: ***Marion McKinley Jr.***

Alan E. Algustyniak
R. Jacque Bruninga
George W. Dietrich
Glen W. Gauthier
Donald R. Lovelace
Raymond L. Mason
William H. Moore
Vernon W. Reynolds
Dale S. Russell

Support Staff: ***Ralph W. Caldwell***
Wanda N. Canillas
Dale M. Chase
Oralio E. Carza
Delia A. Heilig
Kyle P. Lemmons
Jack M. Mann
Rennie Simpson
Deborah C. Williams

INTRODUCTION

The ISI ARPANET TENEX service facility is operated as a research and service center in support of a broad set of ARPA projects. It currently services more than 800 users, 95 percent of whom access the facilities via the ARPANET from locations extending from London, England to Hawaii. All facilities systems are available to all users, whether they are connected through the ARPANET either locally or remotely.

The facility consists of four Digital Equipment Corporation (DEC) PDP-10 central processors (one KI-10 and three KA-10s), Bolt Beranek and Newman (BBN) virtual memory paging boxes, large-capacity memories, on-line swapping and file storage, and associated peripherals (see Figure 8.1). All systems presently run under control of the TENEX operating system (developed by Bolt Beranek and Newman), which supports a wide variety of simultaneous users.

HARDWARE

New hardware acquired in the past year as part of a general upgrading effort includes two additional DEC PDP-10 central processors and BBN virtual memory paging boxes, an additional 768K words of Ampex high-speed memory, ten CALCOMP 230 disk

drives that have more than doubled the previous on-line swapping and file storage capabilities, two additional Systems Concepts channels and a new CALCOMP 1040A/345 magnetic tape system. Figure 8.2 shows the current ISI service facility configuration. Note that none of the central processors, the KA-10s nor the KI-10, operate in dual processor mode. Instead, the main goal of having the several systems is to provide a significant increase in the availability of the ISI primary machine, system A. Thus if one of the systems designated as a primary machine crashes, or is down for hardware/software maintenance or development, then one of the other systems may be started as a primary machine and service continued after a brief (normally 15 minutes) interruption to switch the file storage media.

Also included within the TENEX service facility are one BBN H-516 Interface Message Processor (IMP), one BBN H-316 Terminal Interface Processor (TIP), one DEC PDP-11/40 and Xerox Graphics Printer (XGP), one DEC PDP-11/45 and SPS-41 Signal Processing System (configured as a speech processor), one Multi-Lingual Processor (MLP-900) and several associated peripheral devices such as disk, drums, memory, special ISI developed interfaces, TTY's etc.

SOFTWARE

The demand for ISI's computer cycles far exceeded the available supply for most of the year. Means were needed to reduce the load on the system and to restrict access of designated users, as specified by ARPA and ISI management. During the year a concentrated effort was made to insure that all of the ISI TENEX service machines provided the same level of systems software, i.e., pie slice scheduler, file management, etc., and that all subsystems were updated to correspond to the latest release. This is a continuing effort and once accomplished will allow easier ongoing software maintenance of the ISI TENEX service systems. We also have provided load-leveling across the machine in conjunction with IPTO to assure reasonable response and greatly expanded system utilization.

SUPPORT PERSONNEL

To properly support the new load of four complete TENEX systems ISI has hired additional systems programmers, a complete new staff of computer service engineers in order to perform our own hardware maintenance, and additional full-time and part-time operators. ISI presently provides seven-day-a-week, twenty-four-hour-a-day operator, software, and hardware support of the TENEX service facility. At least one operator is physically on-site at all times, and the systems programmers and computer service engineers are either physically on-site or are scheduled for one-hour on-call service. The addition of the computer service engineers to our staff now provides ISI the capability of performing complete repair, maintenance, and service of all computers and related peripheral equipment within the facility, thus eliminating service contracts with several different equipment vendors. See Fig. 8.3.

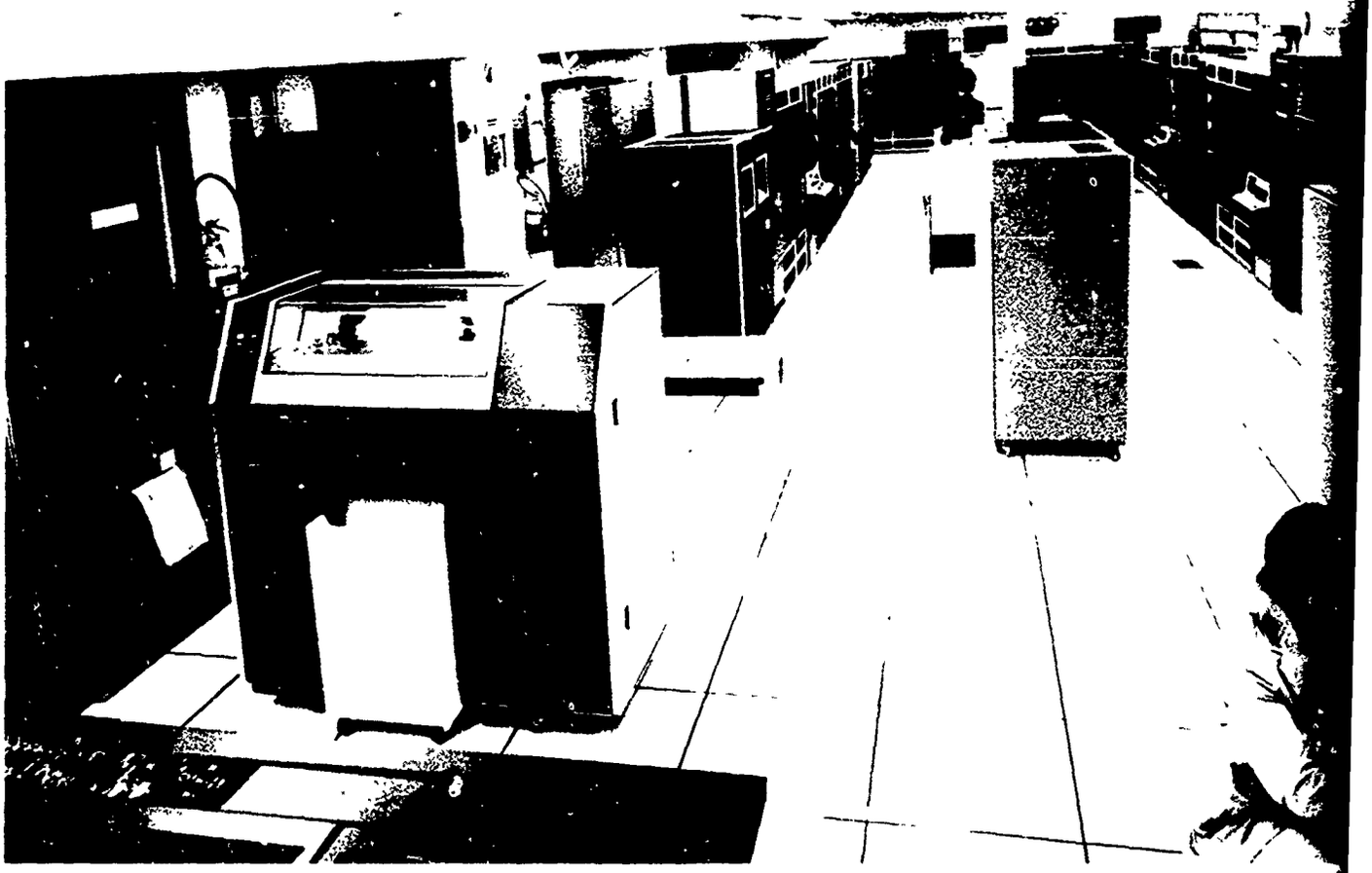

Figure 8 1 Composite photograph o

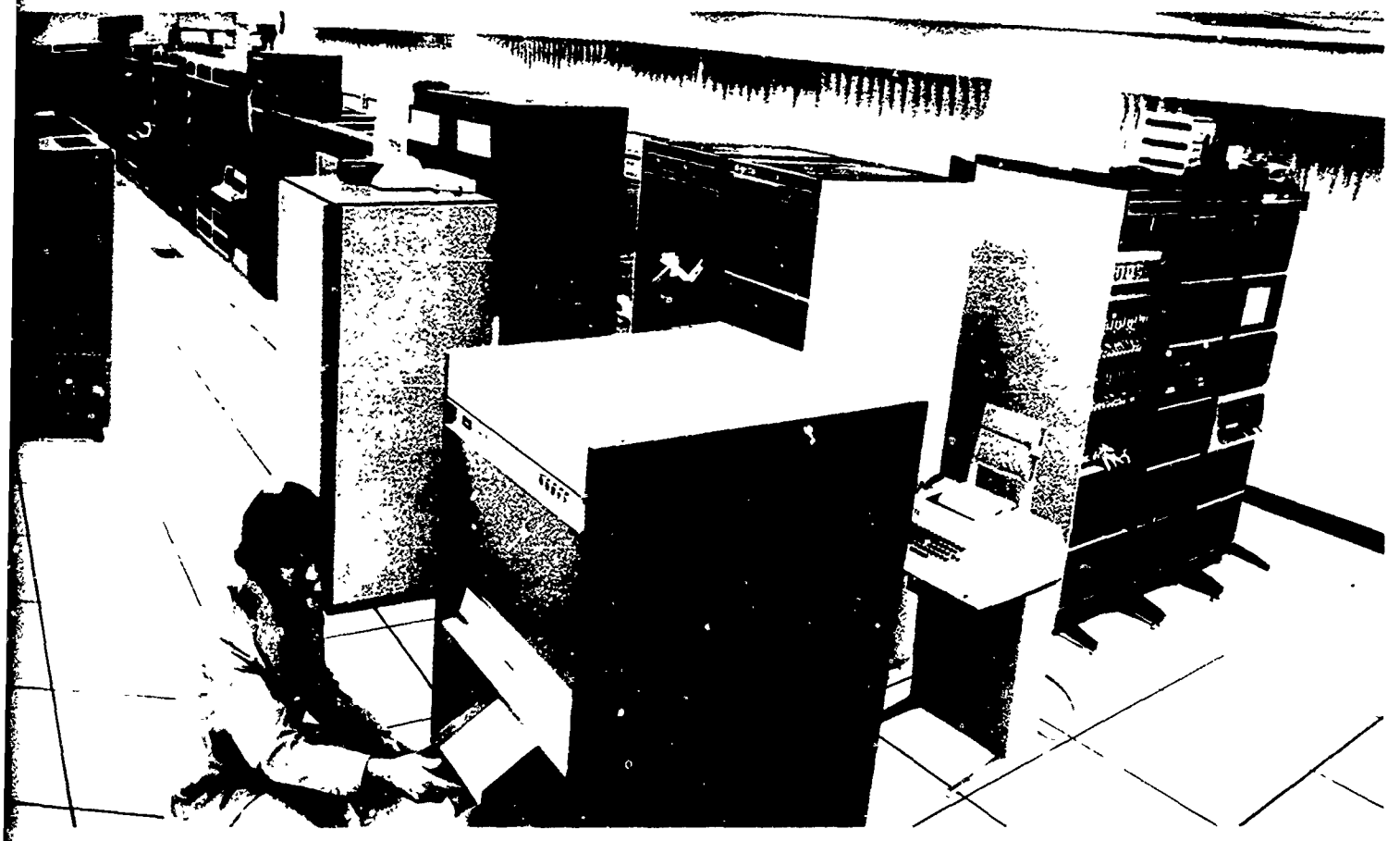

Photo by Marti Coale

.1 Composite photograph of computer room.

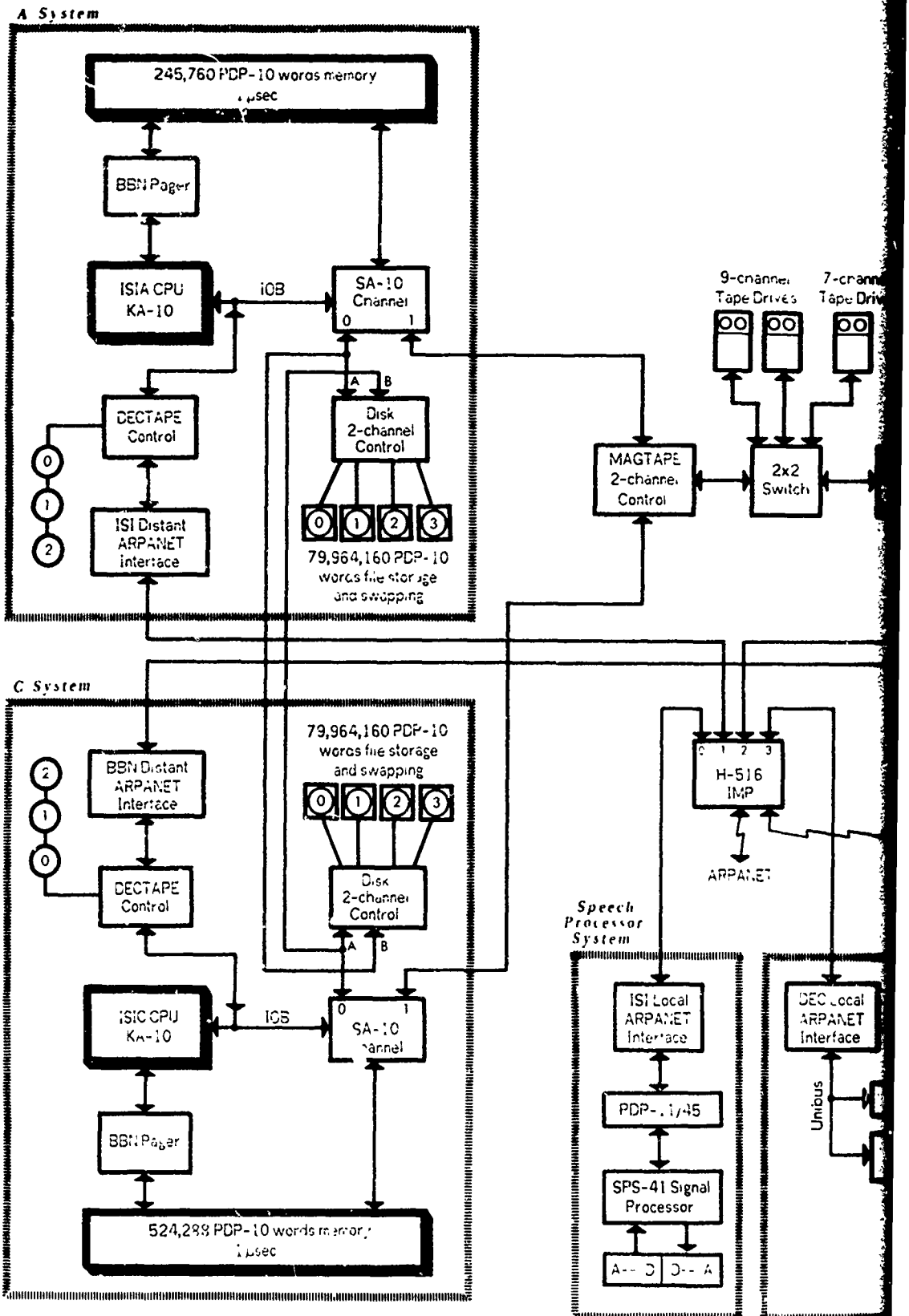

Figure 8.2 Diagram of ISI ARPANET TENEX serv

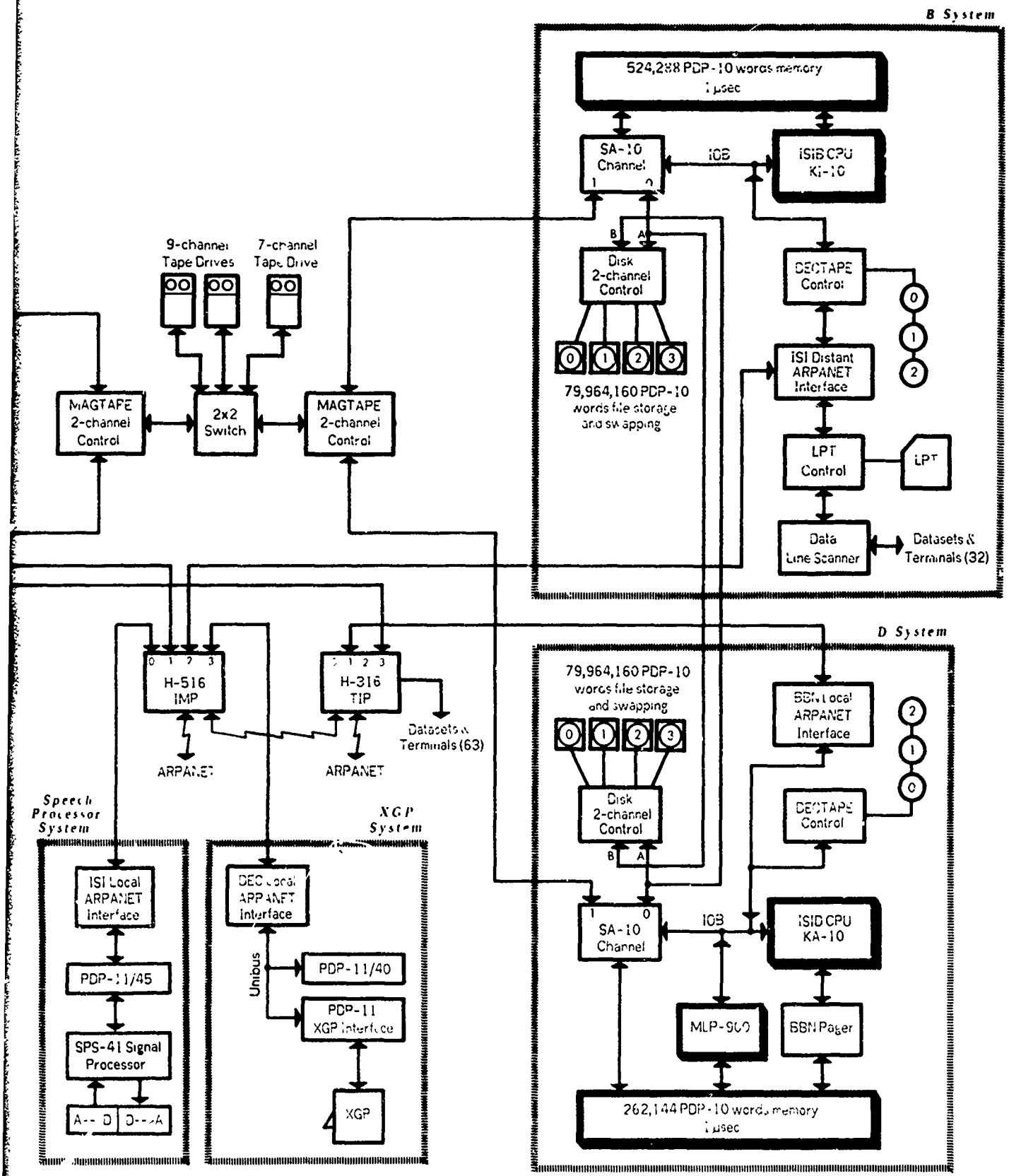

Diagram of ISI ARPANET TENEX service facility.

PERFORMANCE STUDIES

During the past year it was observed that as the load average of a particular system increased, the overall performance and response time of the system drastically deteriorated. Furthermore, upon being assigned the task of providing NLS service on one of our ISIC-KA-TENEX machines, it became quite obvious that (because of the large working set of pages required by NLS) ISI could not provide high-quality service on this machine as it was then configured. It was then decided to undertake performance studies and measurements on a system configured with twice the memory capacity (512K words) of any existing TENEX system. To facilitate these measurements, both hardware and software modifications to the former system were required, and an additional 256K words of memory had to be obtained. Once this was accomplished performance measurements were made by ISI in-house staff and by members of the NLS support group at Stanford Research Institute. All performance measurements on this system were compared to identical measurements made on several other operational TENEX systems via the ARPANET. After all results were compiled and evaluated (and after many congratulatory comments from on-line users) it was concluded that a 512K word TENEX system is the best cost-performance system. ISI presently provides the only 512K word TENEX service to ARPA network users on system C. It is believed that this is a first for a DEC KA-10 processor to run an operational system configured with more than 256K words of memory.

Photo by Marti Coale

Figure 8.3 Computer room operator console area.

RELIABILITY

To provide required hardware/software preventive and/or corrective maintenance of the equipment, ISI as in the past will continue scheduling each of the TENEX systems as "out of service" (unavailable to users) for seven contiguous hours each week. The remaining 161 hours of each week are intended to be devoted entirely (100%) to user service. It is expected that the actual long-term up-time attained during the past year will continue to be greater than 98% (on an 161-hour-per-week basis) for each system.

LOCAL PROJECT SUPPORT

The TENEX facility has been utilized extensively in support of local projects. The staff makes use of all of the available standard subsystems (e.g., editors, compilers, assemblers, and utilities). Additionally, staff members have written subsystems and utilities in support of their own projects. The facility also supports less frequently used subsystems at the special request of users (e.g., PDP-11 cross assemblers and the DECUS Scientific Subroutine Package).

Monitor modifications to support the MLP-900 have been developed and verified. These modifications allow basic processor-to-processor communication through both the input/output (I/O) and memory buses.

PUBLICATIONS

Anderson, Robert H., ***Programmable Automation: The Future of Computers in Manufacturing***, ISI/RR-73-2, March 1973; also appeared in ***Datamation***, Vol. 18, No. 12, December 1972, pp. 46-52.

---, and Nake M. Kamrany, ***Advanced Computer-based Manufacturing Systems for Defense Needs***, ISI/RR-73-10, September 1973.

Balzer, Robert M., ***Automatic Programming***, ISI/RR-73-1 (draft only).

---, ***Human Use of World Knowledge***, ISI/RR-73-7, March 1974.

---, ***Language-Independent Programmer's Interface***, ISI/RR-73-15, March 1974; also appeared in ***AFIPS Conference Proceedings***, Vol. 43, AFIPS Press, Montvale, N. J., 1974.

---, Norton R. Greenfeld, Martin J. Kay, William C. Mann, Walter R. Ryder, David Wilczynski, and Albert L. Zobrist, ***Domain-Independent Automatic Programming***, ISI/RR-73-14, March 1974; also appeared in ***Proceedings of the International Federation of Information Processing Congress***, 1974.

Bisbey, Richard L., and Gerald J. Popek, ***Encapsulation: An Approach to Operating System Security***, ISI/RR-73-17, December 1973.

Ellis, Thomas O., Louis Gallenson, John F. Heafner, and John T. Melvin, ***A Plan for Consolidation and Automation of Military Telecommunications on Oahu***, ISI/RR-73-12, June 1973.

Kamrany, Nake M., ***A Preliminary Analysis of the Economic Impact of Programmable Automation Upon Discrete Manufacturing Products***, ISI/RR-73-4, October 1973.

London, Ralph L., Shigeru Igarashi, and David C. Luckham, ***Automatic Program Verification I: A Logical Basis and Its Implementation***, ISI/RR-73-11, May 1973; also appeared in ***Artificial Intelligence Memo 2000***, Stanford University, May 1973.

Oestreicher, Donald R., ***A Microprogramming Language for the MLP-900,*** ISI/RR-73-8, June 1973; also appeared in the Proceedings of the ACM Sigplan Sigmicro Interface Meeting, New York, May 30-June 1, 1973.

Richardson, Leroy, ***PRIM Overview,*** ISI/RR-74-19, February 1974.

Heafner, John F., ***A Methodology for Selecting and Refining Man-Computer Languages to Improve User's Performance,*** ISI/RR-74-21, September 1974.

Good, Donald I., Ralph L. London, and W. W. Bledsoe, ***An Interactive Program Verification System,*** ISI/RR-74-22, November 1974.

Tugender, Ronald, and Donald R. Oestreicher, ***Basic Functional Capabilities for a Military Message Processing Service,*** ISI/RR-74-23, May 1975.

Abbott, Russell J., ***A Command Language Processor for Flexible Interface Design,*** ISI/RR-74-24, February 1975.

Rothenberg, Jeff, ***An Intelligent Tutor: On-Line Documentation and Help for A Military Message Service,*** ISI/RR-74-26, May 1975.

---, ***An Editor to Support Military Message Processing Personnel,*** ISI/RR-74-27, June 1975.

Carlstedt, Jim, Richard L. Bisbey II, and Gerald J. Popek, ***Pattern-Directed Protection Evaluation,*** ISI/RR-75-31, June 1975.

Heafner, John F., ***Protocol Analysis of Man-Computer Languages: Design and Preliminary Findings,*** ISI/RR-75-34, July 1975.

TECHNICAL MANUALS AND SPECIAL REPORTS

Annual Technical Report, May 1972 - May 1973, ISI/SR-73-1, September 1973.

A Research Program in the Field of Computer Technology, Annual Technical Report, May 1973 - May 1974, ISI/SR-74-2, July 1974.

Gallenson, Louis, Joel Goldberg, Ray Mason, Donald Oestreicher, Leroy Richardson, ***PRIM User's Manual,*** ISI/TM-75-1, April 1975.

COLLOQUIA

July	John Pickens, The PLATO System and the PLATO Terminal
August	Bill Mann, ISI, Multi-Stream Editor Design
	Jim Levin, UCSD, Aspects of Diagnosis
	John Burger, SDC, Conceptual Processing of English
	Richard Hart, University of Connecticut, A System for Answering the Questions of Beginning Lisp Students
	Warren Teitelman, PARC, New Additions to Lisp
September	Bob Balzer, ISI, AP at IBM
October	Larry Fagan, ISI, Modifications to the Verification System
	Ron Tugender, ISI, XED...Or Happiness is a Warm Text Editor That Cares
December	Jim King, IBM, A New Approach to Program Testing
January	Carl Hewitt, MIT, The Programmer's Apprentice
	Sue Gerhart, Duke University, Test Data Selection
	J. T. Schwartz, Courant Institute, Automatic Data Structure Choice
	Jerry Shelton, University of Wisconsin, A Practical Model for Programming Language Semantics
	Jim Levin, ISI, Fisher's Theory of Control Structure

February Jim Carlisle, ISI, Human Communication in Teleconferencing

Jim Moore, ISI, Systematic Methods for Observing and Encoding Group Interactions

Jim Carlisle, ISI, Applications and Taxonomy of Teleconferencing

March Steven Boies, IBM, Speech Filing Systems

John Gould, IBM, Psychological Studying of Program Querying by Non-Programmers

Larry Roberts, Telnet, Telnet's Plans

Eric J. Neuhold, University of Stuttgart, On Correctness Proofs for Command Programs

Giorgio P. Ingargiola, Caltech, Towards a System with Specialized Programming Knowledge

Jerome Elkind, PARC, Review of Current PARC Research

Larry Miller, ISI, Mathematical Pattern Recognition

David Patterson, UCLA, Verification of Microprograms

April Jim Carlisle, ISI, Human Communication in Teleconferencing

Dave Fisher, Institute for Defense Analyses, A Time Linear Bounded Work Space Copying Algorithm/Current Progress on the Definition of a Common DoD Higher Order Language

Richard Johnsson, Carnegie-Mellon University, Register Allocation Optimization

Phillip Mason, Carnegie-Mellon University, The Design of Programs Asynchronous Multiprocessors

Axel Van Lamsweerde, MBLE Research Labs, Brussels, Correctness of Parallel Processes

May

Hanan Samet, Stanford AI Lab, Automatically Proving the Correctness of Optimized Code

Bill Mann, ISI, Recent Progress of Dialogue Modelling

Mark Stickel, Carnegie Mellon University, Incompleteness Aspects of Artifical Intelligence Languages

Bill Mann, ISI, Why Things are So Bad for the Computer-Naive User

Stephen N. Zilles, IBM, Data Algebra

James Griesmer, IBM Yorktown Heights, The SCRATCHPAD System for Symbolic Mathematical Computation

June

Bill Wulf, Carnegie-Mellon University, The ALPHARD Programming Language

Dave Musser, ISI, An Algebraic Evaluator for Conditional Expressions

DOCTORAL THESES

Completed

John F. Heafner, ***Design of Application-oriented Languages by Protocal Analysis***, 1975

Robert W. Lingard, ***A Representation for Semantic Information Within an Inference-making Computer Program***, 1975

David Wilczynski, ***A Process Elaboration Formalism for Program Writing and Analysis***, 1975

Martin D. Yonke, ***A Knowledgeable Language-Independent System for Program Construction and Modification***, 1975

In Progress

Donald S. Lynn, ***Automatic Program Verification: Compiler Proofs***